The d^2-Tree: A Hierarchical Representation of the Squared Distance Function

S. Leopoldseder [1], H. Pottmann [1], H. Zhao [2]
[1] Institute of Geometry, Vienna University of Technology, Vienna, Austria
[2] Department of Mathematics, University of California, Irvine, U.S.A.

Abstract

The d^2-tree as presented in this paper is an octree data structure which stores in each of its cells a local quadratic approximant of the squared distance function of a geometric object M. This adaptive data structure has large cells in the far field of the geometric object M whereas small cells are used in the near field of M and also in areas where the squared distance function of M is not differentiable, i.e., at the medial axis of M.

Our proposed data structure is appropriate for geometric optimization problems, like registration or active surface approximation, where the solution to the problem is found iteratively with a Newton-type method. An example for active surface approximation is briefly presented.

1 Introduction

The distance function of a curve or surface M assigns to each point $\mathbf{x}$ of the embedding space the shortest distance $d(\mathbf{x})$ of $\mathbf{x}$ to M. Since d is not differentiable at M one often uses the signed distance function, which agrees with d up to the sign. It is well defined for a closed object and takes on different signs inside and outside the object, respectively. In the following, we will just speak of the distance function for both the signed and the unsigned version. A variety of contributions deals with the computation of this function and its use in various algorithms of geometry processing.

In many cases the computation aims towards the singular set of the function, i.e., towards points where the function is not smooth. Those points lie on the cut locus (medial axis, skeleton) of the input shape. Recently, a survey of this topic has been given by Choi and Han [3].

The distance function is also the viscosity solution of the eikonal equation $\|\nabla f(\mathbf{x})\| = 1$. Its numerical computation is not trivial because it is a nonlinear hyperbolic equation and an initially smooth front may develop singularities (shocks) as it propagates. Precisely the latter belong to the cut locus. The computation of viscosity solutions with the level set method of Osher and Sethian [12, 19] proved to be a very powerful approach. Whereas earlier implementations have been of complexity $O(N \log N)$, in the number N of grid nodes, H. Zhao [21] recently presented an algorithm of complexity $O(N)$. This algorithm is heavily used in the present work.

The level sets of the distance function of M are the offsets of M, which are of particular importance in Computer Aided Design (see e.g. [13]) and also appear in mathematical morphology [18]. Results on the distance function and its application to shape interrogation for CAD/CAM may also be found in the recent book by Patrikalakis and Maekawa [13].

The distance function or other functions, which vanish exactly at the object of interest, have also received great attention within Computer Graphics (see, e.g. [5, 7, 10]) In particular, adaptively signed distance fields (ADF's) [5] proved to be a versatile and unifying representation with many applications. ADF's are closely related to the present work.

Our work originates from the solution of various geometric optimization problems. One example are registration problems. Given, for instance, a cloud of measurement points and a CAD model M of an object, one aims at moving the point cloud, viewed as elements of a rigid body, as close as possible to M. Typically, one computes a final position of the point cloud such that the sum of squared distances of points in the cloud to M is minimial. The main solution is the well-known iterative closest point (ICP) algorithm [1]. In each iteration, ICP moves the points towards their closest points on the CAD model; per step one has to solve an eigenvalue problem. There are many extensions of this algorithm, which still compute corresponding points in some way [1, 17]. Recently, we suggested to let the point cloud just flow in the squared distance field

of the CAD model. Using kinematics and local quadratic approximants of the squared distance function of M, we showed that this requires just the solution of a linear system in each step of an iterative algorithm. Moreover, it has been shown how to extend the algorithm to simultaneous correspondence-free registration of N systems (point clouds or surfaces) and how to incorporate some correspondences, if they are known [16].

The moving point cloud can also undergo some deformation. In particular, it can be a cloud of points on a deforming B-spline surface. This results in an active contour approach to surface approximation, which will be discussed in section 4. Again, unlike previous approaches, where correspondences are used to drive the active model [9], we just use local quadratic approximants of the squared distance field of the model shape M, which shall be approximated.

In our work we have so far used differential geometric knowledge, which ties local quadratic approximants of the squared distance function $d^2(\mathbf{x})$ of a surface M to the curvature behavior of M [14]. We have been missing an appropriate structure, from which we can quickly retrieve the necessary information about $d^2(\mathbf{x})$. This gap shall be filled in the present paper.

We present an efficient algorithm for the computation of an octree, denoted as d^2-tree. Each of its cells carries a quadratic function

$$f(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x} + c, \quad A \text{ symmetric}, \mathbf{x} \in \mathbb{R}^3, \tag{1}$$

which is an appropriate approximation to the squared distance function $d^2(\mathbf{x})$ for all points $\mathbf{x}$ of the cell. It is important that the quadratic function $f(\mathbf{x})$ is not restricted to the points $\mathbf{x}$ within the cell but it is a function on the whole embedding space $\mathbb{R}^3$, see Fig. 4 for an illustration of the planar case.

In view of our applications, less accuracy and thus larger cells are used in the far field of M, whereas we have smaller cells close to M. This is a close relation to ADF's [5], but ADF is just a method for adaptive rendering or representation of a geometric object with a given distance field. Our interest lies in dynamic deformation or control of an object. Also ADF does not address the issue about how to compute the distance field efficiently and adaptively.

One main difference to ADF's is that we do not store just the values of the function d^2, but coefficients of quadratic approximating functions. We store more information in each cell but this is not inefficient in terms of storage requirement because we need much less cells to store the same information on the local behavior of d^2.

Of course the computation of the coefficients A, b, c of $f(\mathbf{x})$ is an extra effort compared to the generation of an ADF. In an application where one is satisfied to have fast access to the distance of certain sample points to the geometric object, an ADF is fine. In geometric optimization problems, however, which are solved with a Newton-type method we need in each iteration local quadratic approximants $f(\mathbf{x})$ for a large number of sample points. There it is favorable to precompute these local quadratic approximants and retrieve them quickly from the d^2-tree during the optimization procedure.

Another difference of our d^2-tree to an adaptively sampled distance field (ADF) is that the size of the cells is not only guided by the distance to M or the local geometry of M. It is also guided by the approximation error between the quadratic function in a cell and the function d^2. At the medial axis of M, e.g., the function d^2 is not differentiable and there is no quadratic function which will locally approximate d^2 well. Thus we obtain smaller cells near the medial axis of M which is a very important property in the applications we have in mind.

2 General Outline

The algorithm will proceed through the following steps which are explained in more detail in the next section. In the following we consider the 3-dimensional setting. The 2-dimensional case is completely analogous.

1. Start with a cube which encloses the geometric object M and those parts of the surrounding 3-space where we are interested in the distance function of M. Subdivide the cube into

its eight sub-cubes. Each of these children (together with neighboring cubes, as specified in Sec.3.1) is subdivided again if it contains part of the geometric object M. Iterate this procedure until the cell size at the finest level is small enough. We obtain an octree data structure `AuxTree` with small cells near M and larger cells further away from M.

2. We assign to the corner points of the resulting cubes of `AuxTree` their distance values to M. For memory reasons we store together with each cube only the distance of its 'lower left' corner point to M. Under the 'lower left' corner point we understand the one corner point of the cube which has minimal x-, y-, and z-coordinates.

   Since it is too time-consuming to evaluate exact distances we use an adapted version of Zhao's sweeping algorithm[21]: We start with the cubes at the finest refinement level of `AuxTree`. Only for those cubes that contain part of M we compute the exact distance value of their 'lower left' corner point to M. Note that in case of M being a triangulated surface, we really take the exact distance values, i.e., the shortest distance to the triangular faces of M. These exact distances are *not* approximated simply by the shortest distance to the vertices of the triangulated surface M.

   Sweeping through the cubes at the finest refinement level, their distance value is updated via Zhao's method. After finishing the finest level of cells the sweeping algorithm is applied to the next coarser level of cubes. The initialization of the distance values on the coarser level is obtained from the finer level.

   After iterating through all refinement levels we have an approximate distance value $d(\mathbf{p})$ for the 'lower left' corner $\mathbf{p}$ of each cell of `AuxTree`. We square this value and obtain a squared distance function d^2 which is defined on the multi-level grid of the cubes' corner points.

3. Finally, we generate the d^2-tree, denoted by `D2Tree`, which is the octree data structure which will contain the quadratic approximants, see Equ. (1), of the squared distance function of M. We begin with a cube at the coarsest level and compute a least square fit of all values $d^2(\mathbf{p})$ to data points $\mathbf{p}$ that lie in this cube. If the residuals of this fit are not satisfactory we subdivide the cube and iteratively compute a least square fit for each of the subcubes.

3 The algorithm

In the following we assume that the geometric object M is a triangulated surface, see e.g. Fig. 1,

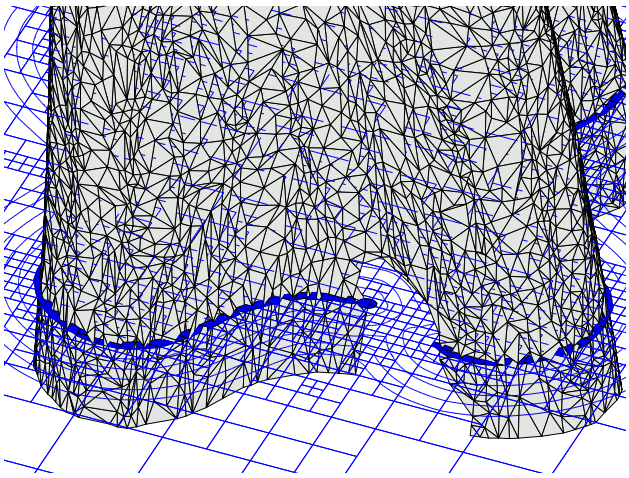

Figure 1: Surface M and a planar slice through the octree data structure `D2Tree` where local quadratic approximants of the squared distance function d^2 of M are stored.

which shows a detail of an architectural design model, see Fig. 7, left, of section 4.1. We are

interested in local quadratic approximants $f(\mathbf{x})$, see Equ. (1), of the squared distance function $d^2(\mathbf{x})$ of M. For purposes of instruction we will in the following visualize only planar slices of the occurring octree data structures `AuxTree` and `D2Tree`.

3.1 The octree cell structure `AuxTree`

First we construct an octree cell structure `AuxTree`. Each cell will store the squared distance d^2 of its 'lower left' corner point. The name of this structure is motivated by the fact that this octree is an auxiliary data structure for the construction of the octree cell structure `D2Tree` which will store quadratic functions $f(\mathbf{x})$ as described above.

At the coarsest level (level 0) of the structure `AuxTree` we initialize the largest cube C^0. Its size is chosen such that the cube encloses M and the complete region of $\mathbb{R}^3$ where the distance function of M is of interest.

We subdivide C^0 and again subdivide its eight children $C_i^1, i = 0, \ldots, 7$ and obtain 8^2 cubes C_j^2 at level 2. Note that if we decide to subdivide a cube of `AuxTree` at level L we will instantly subdivide it into its 8^2 subcubes at level $L + 2$. Jumping from level L directly to level $L + 2$ has an advantage lateron with the adapted sweeping algorithm of Zhao, see section 3.2. There it is more efficient to have larger blocks of cells of the same size, i.e., we prefer larger blocks of cells in the same level. The finest level will be denoted by L_{max}. The level L_{max} is chosen such that the size of its cubes meet the precision requirement of the application.

In order to decide which cubes of `AuxTree` are hit by at least one triangle of the triangulated surface M we run through the list of triangles once. For each triangle we proceed as follows: Beginning with level $L = 2$, we mark the cells at level L that are hit by the current triangle, and we subdivide them to level $L + 2$. Only these resulting cells of level $L + 2$ need to be tested on intersection with the current triangle which then results in further subdivision to level $L + 4$. In this way we continue to the finest level L_{max}, and continue with the same procedure for the next triangle of M.

After tracing all triangles of M we obtain an octree `AuxTree` where the cells in a very narrow band around M are subdivided to finer cells. Now this narrow band will be extended outwards: For each level L certain cells C_j^L are already subdivided to level $L + 2$. Now all the neighboring cells of C_j^L are also subdivided to level $L + 2$. Different neighborhood definitions are possible, for instance the 1-ring of direct neighbors, i.e., the 26 cubes that share at least a corner point with the central cube. Alternatively one could take the 2- or 3-neighborhood of the cubes C_j^L.

See Fig. 2 for an illustration of a planar slice through the octree `AuxTree` that was constructed for the triangulated surface of Fig. 1. The position of the planar slice is indicated in Fig. 1 also. A 3-neighborhood definition was chosen for this example.

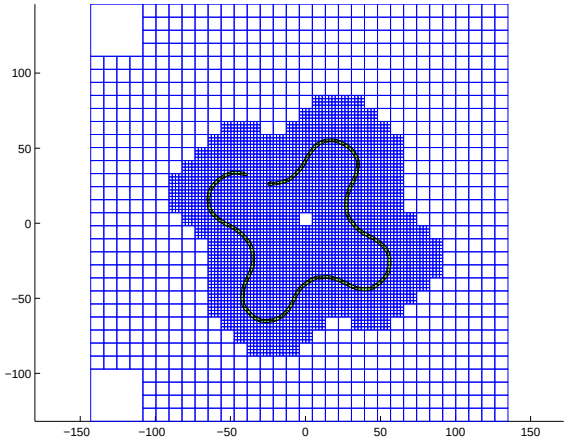

Figure 2: Planar slice through octree `AuxTree`. Small Cells appear in the neighborhood of the surface M.

3.2 Multilevel version of Zhao's sweeping algorithm

For each cube C in `AuxTree` we want to obtain a distance value of its 'lower left' corner point $\mathbf{p}_C$ to M. An exact distance computation will be performed only for the cells of the finest level which are close to M. This exact distance information will be extended outwards on the remaining grid points $\mathbf{p}_C$ via Zhao's sweeping algorithm [21]. This algorithm exploits the fact $\|\nabla d(\mathbf{x})\| \equiv 1$ for the distance function $d(\mathbf{x})$ of M.

The distance values of all the cubes of `AuxTree` are pre-set to infinity. For the moment we only look at the cubes of the finest level. For an initialization of distance values we run through the list of triangles of M. If a triangle hits a cube then the distances of the cubes' corner points to this triangle are computed. The new distance value for these corner points is the minimum of the currently stored distance and the distance to the triangle.

The cubes in `AuxTree` at the finest level form a regular grid. Zhao's sweeping algorithm defines eight sweeping directions in three-space, namely 'x+y+z+', 'x+y+z-', 'x+y-z+', ..., 'x-y-z-'. We explain the notation of the sweeping directions at hand of one example, say 'x+y+z+'. This denotes a sweep in positive x-, then positive y-, then positive z-direction, i.e., an iteration

```
for z=z_min to z_max
   for y=y_min to y_max
      for x=x_min to x_max
         update distance information of p =(x,y,z)
      end
   end
end
```

An upwind type of finite difference approximation is used for the partial differential equation $\|\nabla d(x)\| = 1$ at each grid point so that the distance value at a grid point depends only on its neighboring distance values that are smaller, i.e., distance information propagates from nearby points to far away points. Also the distance value at any grid point is updated only if the newly computed value is smaller that its previous one. Using these eight sweeps with different ordering the distance values at all grid points can be computed efficiently. For details of the numerical algorithms see [21].

In the first step we have now supplied each cube of the finest level L_{max} in `AuxTree` with a distance value to M. In the next step we again apply Zhao's algorithm for the cubes of level $L_{max} - 2$. Some of the cubes of this coarser level are subdivided into cubes of level L_{max}, namely those that are close to M. Those cubes obtain their distance value from their subdivided cubes whose distance value has been previously computed. We again apply Zhao's algorithm, now for the cubes of level $L_{max} - 2$ and obtain distance values for all cubes of this level. We iterate this procedure till we reach the single cube C^0 in level 0. Now we have finished the construction of the octree `AuxTree`, since we have obtained a distance value to our geometric object M for each cube at an even level L.

Let us conclude this section with a focus on an important detail on the implementation of this adaptation of Zhao's sweeping algorithm: Since our cubes in `AuxTree` are organized in an octree, sweeping in the x-, y-, and z-coordinate directions may become rather slow. Thus we sort our cubes in the eight required sweeping orders 'x+y+z+', 'x+y+z-', ..., which are defined above. The sorting is performed in a top-down procedure for all levels $2, 4, \ldots, L_{max}$ before we start the sweeping algorithm. We explain this procedure in more detail at hand of the 'x+y+z+' sweeping direction. The other seven sweeping directions are completely analogous.

Beginning with level $L = 2$ we assume that all cells of level L are already sorted in the correct order and are stored in a sorted list C_j^L: Those cells of level L with minimal z-coordinate $z = z_{min}$ will appear at the beginning of the list C_j^L, sorted in 'x+y+' order. The list C_j^L continues with z-parallel layers of cells, each sorted in 'x+y+' order, up to the last layer of cells with $z = z_{max}$.

Now we want to sort the cells of level $L + 2$ in the same sweeping order 'x+y+z+' and store them in a list C_k^{L+2}. First we delete all cells from the list C_j^L that are not further subdivided to level $L + 2$. The remaining cells in C_j^L are still in the correct 'x+y+z+' order and this order helps us to sort their grandchildren of level $L + 2$. We treat each z-parallel layer of cells in C_j^L

separately, and each of those layers of level L leads to four z-parallel layers in the list C_k^{L+2} of level $L+2$. Again, each of those z-parallel layers within the list C_k^{L+2} must be sorted in 'x+y+' order, which is easy because their corresponding cells in C_j^L are already sorted in the same order.

3.3 Quadratic approximants of the squared distance function

At the current moment we have an octree cell structure `AuxTree` which is composed of cubes that are of small size near the geometric object M and of larger size farther away from M. For all cubes C of `AuxTree` there is stored a distance value d_C which is the distance of the 'lower left' corner point $\mathbf{p}_C$ of C to M. We square all these distances and store d_C^2 for each cell C.

We are now constructing a new octree structure named `D2Tree`. This structure will be similar to `AuxTree` but will store a quadratic function

$$f_Q(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x} + c, \quad A \text{ symmetric}, \mathbf{x} \in \mathbb{R}^3$$

for each cubical cell Q of `D2Tree`. Within the limits of the cube Q the function f_Q shall be a sufficiently close approximation to the squared distance function of M. The function f_Q is not restricted to the cube Q, however, but is defined on the whole embedding space $\mathbb{R}^3$.

We start with the largest cell Q^0 of `D2Tree` which has the same size as the largest cell C^0 of `AuxTree`. The cell C^0 contains a large number of subcells of different sizes. Each of these subcells has a data point $\mathbf{p}$ (its 'lower left' corner point) with a squared distance value $d^2(\mathbf{p})$.

Now we compute a weighted least square fit $f_{Q^0}(\mathbf{x})$ of all the data $d^2(\mathbf{p})$, where $\mathbf{p}$ is contained in Q^0. If the residuals are sufficiently small we accept this quadratic function f_{Q^0}, otherwise we subdivide Q^0 into its eight subcubes $Q_i^1, i = 0, \ldots, 7$ of level 1. For each of the cubes Q_i^1 we again compute a weighted least square fit $f_{Q_i^1}(\mathbf{x})$ of the data $d^2(\mathbf{p})$, where $\mathbf{p}$ is contained in the subcell Q_i^1. This procedure is iterated until we have an adequate quadratic function $f_{Q^L}(\mathbf{x})$ for each cube Q^L of `D2Tree`. If there are not enough data points $\mathbf{p}$ within a certain cube Q^L at level L for a robust estimation of $f_{Q^L}(\mathbf{x})$ we take the quadratic function $f_{Q^{L-1}}(\mathbf{x})$ of its parent cube Q^{L-1}.

Of course there are a lot of data points $\mathbf{p}$ in a coarse cell, e.g. in Q^0. It is very unlikely that all the data $d^2(\mathbf{p})$ may be fitted with one single quadratic function (unless M is a single point or a plane). In order to speed up the computation, say for f_{Q^0}, one will first try to fit only the data $d^2(\mathbf{p})$ to the points $\mathbf{p}$ of the coarse level 2 of `AuxTree`. Should no adequate quadratic function f_{Q^0} exist to this reduced data set, we may already subdivide Q^0 to the next level 1. Otherwise we will gradually add points $\mathbf{p}$ of finer levels $4, \ldots, L_{max}$ of `AuxTree` till either the least square fit f_{Q^0} fails or we end with an adequate function f_{Q^0} approximating all the data.

Figure 3 depicts a planar slice through the cell structure `D2Tree`, corresponding to our example

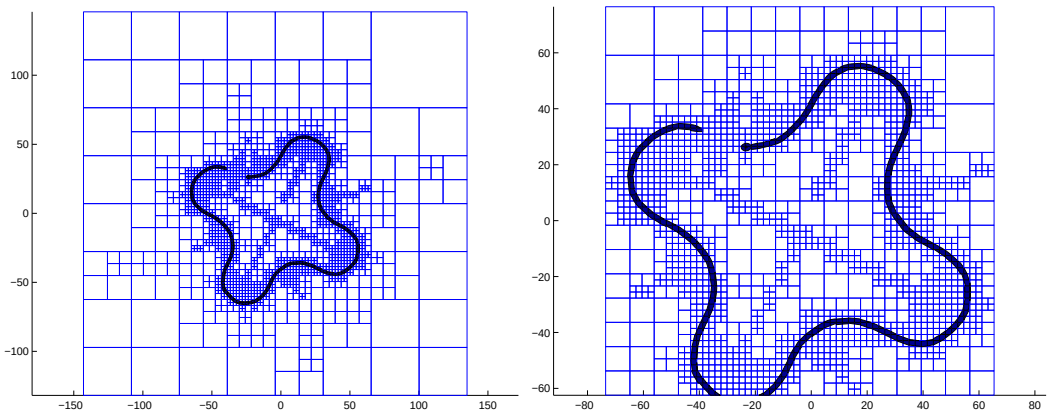

Figure 3: Left: Planar slice through `D2Tree`, Right: Detail. Small Cells appear near the surface M and in the area of the medial axis of M.

model M of Fig. 1. Small cells appear near M and in the area of the medial axis of M. At the

medial axis the squared distance function d^2 to M is not differentiable, thus there exist no 'good' local quadratic approximants of d^2.

Figure 4, left and right, show the levels sets of the quadratic functions $f_{Q_i}(\mathbf{x}), i = 1, 2$ that

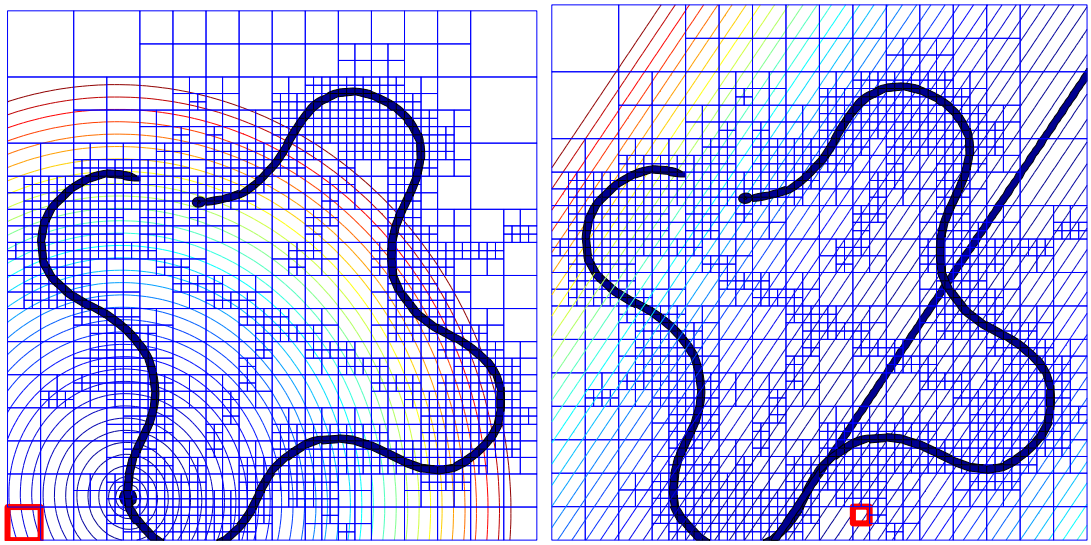

Figure 4: Planar slice through `D2Tree`. The local quadratic approximations $f_{Q_1}(\mathbf{x})$, $f_{Q_2}(\mathbf{x})$ stored in the marked cells Q_1 (left) and Q_2 (right) are visualized by some of its level sets. The locus of minimum values of $f_{Q_i}(\mathbf{x})$ (a point for f_{Q_1} and a line for f_{Q_2}) are also depicted.

are stored in the marked cells Q_i of `D2Tree`. The illustrations are again from a planar slice of the 3-dimensional octree structure `D2Tree`. The locus $\mathbf{x}$ where the function $f_{Q_1}(\mathbf{x})$ of Fig. 4, left, takes its minimum value is also marked and this points lies close to the surface M. If $f_{Q_1}(\mathbf{x})$ were not determined numerically, but were the exact local quadratic Taylor approximant of the squared distance function of M in the midpoint $\mathbf{m}$ of the marked cell, then $f_{Q_1}(\mathbf{x})$ would have its minimum value exactly at the normal footpoint of $\mathbf{m}$ to M, and this minimum value of f_{Q_1} would be zero.

Only non-negative quadratic functions $f_Q(\mathbf{x})$ are useful for the applications we have in mind. Thus, if there are negative eigenvalues of the symmetric matrix A in equation (1) we replace the function $f_Q(\mathbf{x})$ by a function $\bar{f}_Q(\mathbf{x})$ where the negative eigenvalues of A are replaced by zero. Figure 4, right, shows the levels sets of such a quadratic function $\bar{f}_{Q_2}(\mathbf{x})$. If one of the eigenvalues is replaced by zero, the minimum value of $\bar{f}_Q(\mathbf{x})$ will be obtained at the points of a line. If two of the eigenvalues are replaced by zero, the minimum value of $\bar{f}_Q(\mathbf{x})$ will be obtained at the points of a plane.

Fig. 5 gives the level sets of different quadratic functions $f_Q(\mathbf{x})$ where each function is plotted only within the cell Q where it is stored. Each of the quadratic functions is smooth in $\mathbb{R}^3$, but the collection of the functions $f_Q(\mathbf{x})$, each restricted to its cell Q, is not smooth along the cell boundaries. This can be also seen in Fig. 6 where the quadratic functions $f(\mathbf{x})$, restricted to their cells, are plotted over the planar domain. This axonometric view shows only a small part (the lower left region) of Fig. 5.

4 Applications

4.1 Approximation with active B-spline surfaces

An efficient approach to various approximation problems for curves and surfaces are *active contour models*, which are mainly used in Computer Vision and Image Processing. The origin of this technique is the seminal paper by Kass, Witkin and Terzopoulos [8], where a variational formulation of

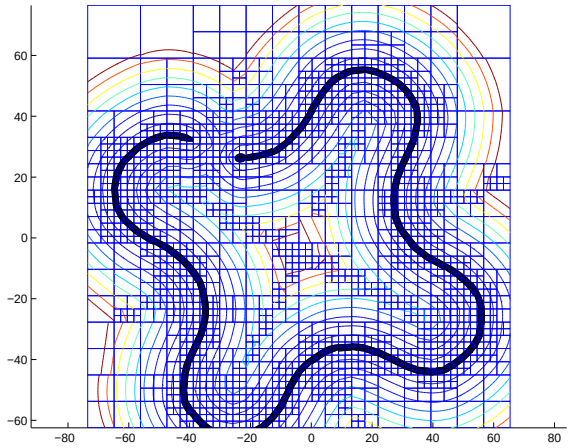

Figure 5: Planar slice through `D2Tree` and level sets of the local quadratic approximations of squared distance function of the surface M.

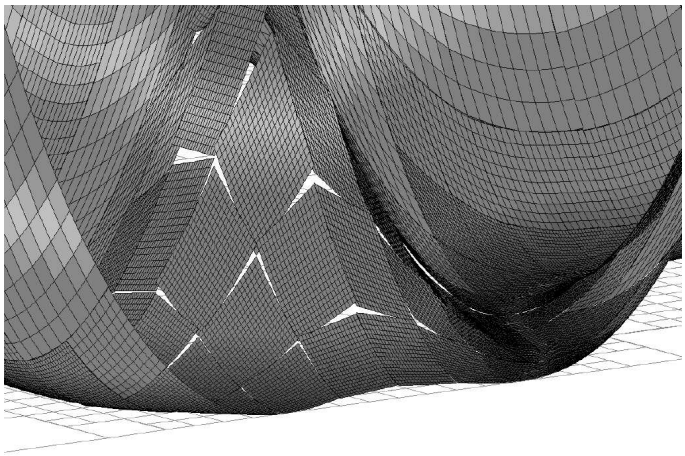

Figure 6: Planar slice through `D2Tree` and the graph of the local quadratic approximants of the squared distance function of the surface M.

parametric curves, called *snakes*, is presented for detecting contours in images. There are various other applications and a variety of extensions of the snake model (see e.g. [2, 11]). Recently we have developed an active contour type strategy [15] for approximating a point cloud or a surface in any representation ('model shape') by a B-spline surface or another surface type which can be written as linear combination of bivariate basis functions

$$\mathbf{x}(u,v) = \sum_{i=1}^{n} B_i(u,v)\mathbf{d}_i. \tag{2}$$

This technique is based on local quadratic approximants of the squared distance function to curves and surfaces as we have described them in Sec. 3. For a differential geometric treatment of this topic, see [14]. The surface approximation method proceeds in the following steps:

1. Initialize the 'active' B-spline surface and determine the boundary conditions. This requires the computation of an initial set of control points, the proper treatment of boundaries (e.g. by fixing vertices of a patch) and the avoidance of model shrinking during the following steps.

2. Repeatedly apply the following steps a.–c. until the approximation error or change in the approximation error falls below a user defined threshold:

a. With the current control points $\mathbf{d}_i$, compute a set of points $\mathbf{s}_k = \mathbf{s}(u_k, v_k)$ of the active surface, such that the shape of the active surface is well captured. For each of the points $\mathbf{s}_k$ determine a local quadratic approximant $f_{\mathbf{s}_k} =: f^k$ of the squared distance function to the model shape at the point $\mathbf{s}_k$. We trace our octree cell structure `D2Tree` and find the smallest cell where $\mathbf{s}_k$ is lying in. The quadratic function stored in this cell will be taken as f^k.

b. Compute displacement vectors $\mathbf{c}_i$ for the control points $\mathbf{d}_i$ by minimizing the functional

$$F = \sum_{k=1}^{N} f^k(\mathbf{s}_k^*) + \lambda F_s, \tag{3}$$

where $\mathbf{s}_k^*$ denote the displaced surface points $\mathbf{s}_k^* = \sum_{i=1}^{n} B_i(u_k, v_k)(\mathbf{d}_i + \mathbf{c}_i)$ (which depend linearly on the unknown displacement vectors of the control points). F_s denotes a smoothing functional which shall be quadratic in the unknown displacement vectors $\mathbf{c}_i$. Thus, our goal is to bring the new surface points $\mathbf{s}_k^*$ closer to the model shape than the old surface points $\mathbf{s}_k$. Since the points $\mathbf{s}_k^*$ depend linearly on the unknown displacement vectors $\mathbf{c}_i$ of the control points, both f^k and F are quadratic in the unknowns $\mathbf{c}_i$.

We see that this step requires the minimization of a function F which is quadratic in the displacement vectors of the control points. This amounts to the solution of a linear system of equations.

c. With the displacement vectors from the previous step, update the control points of the active surface.

An important advantage of the new technique is that it is not necessary to deal with the correspondence between points in the parameter domain and the data points. Thus problems where this correspondence is crucial can now be easier handled.

As an example, Fig. 7, left, shows a triangulated point cloud M as the geometric model. The data has been obtained by scanning an architectural design, namely a clay model of a tower. In the lower left part of the model you may note a small gap in the model. This is a local artifact where the laser based data capturing failed. Topologically the surface M is a cylindrical patch. This surface shall be approximated by a B-spline surface which is closed in u-parameter direction, see Fig. 7, right. The initial position of the active B-spline surface was chosen as in Fig. 8, left. The B-spline surface is chosen bicubic, i.e., degree $(3, 3)$, with 24×6 control points.

The control points $\mathbf{d}_i$ of the active B-spline surface, which are not depicted in Fig. 7 and Fig. 8, are iteratively recomputed as described above, such that the B-spline surface 'flows' towards the target shape. Fig. 8, right, shows the final result as an overlay of Fig. 7, left and right.

As a boundary condition, two planes ε_0 and ε_1 have been chosen where the closed boundary curves $v = v_0$ and $v = v_1$ of the active surface patch are lying in. During the iterative surface approximation procedure, the control points corresponding to these two boundary curves are only allowed to move within ε_0 and ε_1, respectively. This side condition is linear in the unknown displacement vectors of the control points. All other control points have their full three degrees of freedom.

5 Conclusion and Future Research

We have presented the d^2-tree, which is an octree whose cells carry local quadratic approximants of the squared distance function d^2 to a geometric object. At hand of surface approximation with active B-spline curves it has been shown that this data structure is very well suited for use in geometric optimization algorithms of the Newton type, where local quadratic approximants of d^2 are necessary. Another application, where this is useful, is registration.

There are many directions for future research. We outline a few of them.

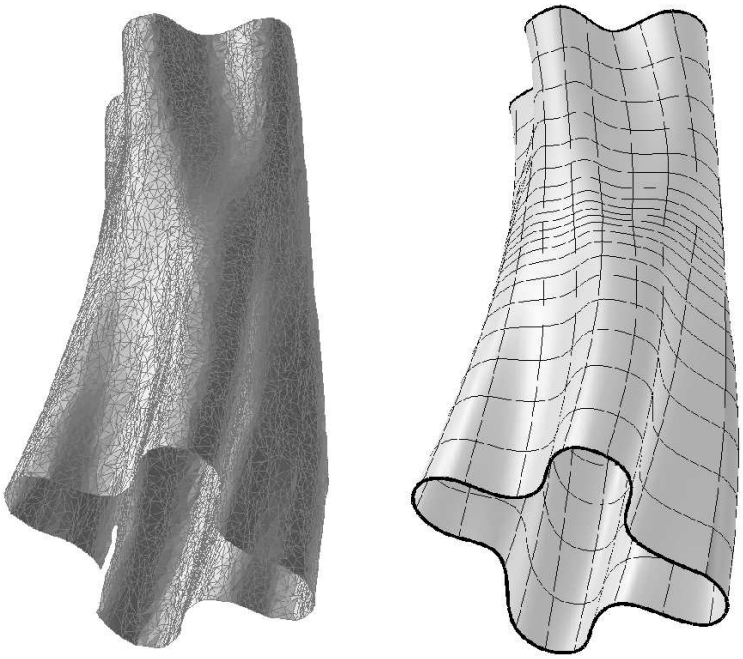

Figure 7: Approximation of a model surface M (triangulated point cloud, left) with a closed tensor product B-spline surface (right).

- A surface computed with the level set method on a grid can be converted into a NURBS surface or a subdivision surface by first converting the level set function into the d^2-tree and then applying the active B-spline model. We can use rational B-splines and optimize the weights as well. There, the unknown displacement vectors are in $\mathbb{R}^4$ and we use a local linearization of the canonical projection $(x_1, x_2, x_3, x_4) \mapsto (x_1/x_4, x_2/x_4, x_3/x_4)$ into $\mathbb{R}^3$. Apart from this modification, the remaining procedure is the same as outlined in this paper.

- Several improvements are possible for the basic active B-spline surface approximation scheme presented here. This includes for instance the (semi-)automatic choice of the start position of the active B-spline surface, or an automated knot insertion and removal process during the optimization process. The latter improvement has been presented by Yang et al. [20] for the curve case but their method can not be applied to the surface case straightforwardly.

- It might be sufficient to store local quadratic approximants of a function, which is not exactly d^2. Therefore, one could try to use the quadric error metrics of Garland and Heckbert [6], which are obtained by summing up squared distances to planar faces of the model M.

- Also in higher dimensions $n > 3$, a d^2-tree makes sense. Examples for applications concern spaces, whose points represent shapes (from $\mathbb{R}^2$ or $\mathbb{R}^3$). A class of learned shapes corresponds to a manifold $M \subset \mathbb{R}^n$. The squared distance function to M, in an appropriately introduced Euclidean metric, can be used to incorporate shape knowledge into data fitting or segmentation algorithms. This would be an extension of work by Cremers, Schnörr and Weickert [4].

Acknowledgements

This work has been carried out as part of the project P16002-N05 of the Austrian Science Fund (FWF).

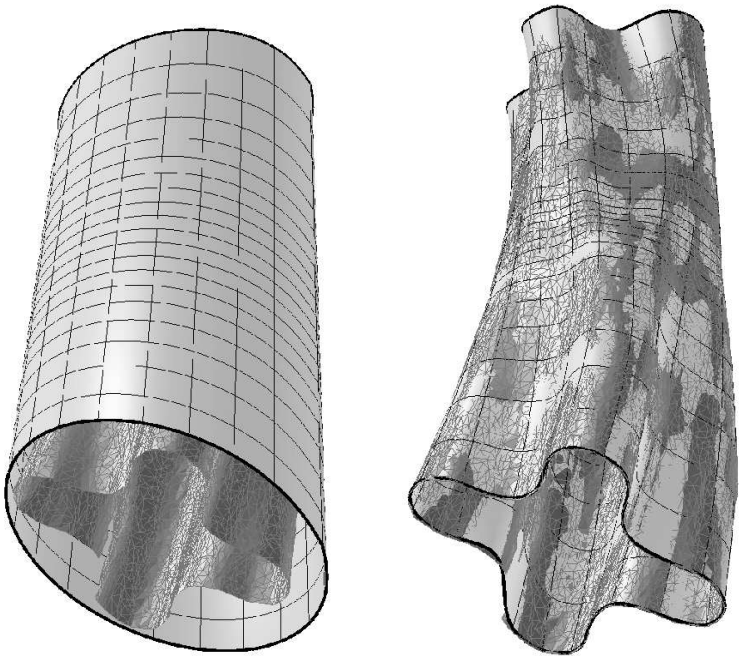

Figure 8: Approximation of a model surface (triangulated point cloud) with a closed tensor product B-spline surface. Left: initial position of active B-spline. Right: final position of active B-spline after 20 iterations.

References

[1] P.J. Besl, N.D. McKay, A method for registration of 3D shapes, *IEEE Trans. Pattern Anal. and Machine Intell.* **14**, pp. 239–256 (1992)

[2] A. Blake, M. Isard, *Active Contours*, Springer, 1998.

[3] H.-I. Choi, C.-Y. Han, The medial axis transform, in: G. Farin, J. Hoschek, M.-S. Kim, eds., *Handbook of Computer Aided Geometric Design*, North Holland, pp. 451–471 (2002).

[4] D. Cremers, C. Schnörr, J. Weickert, Diffusion snakes: combining statistical shape knowledge and image information in a variational framework, In: M. Figueiredo, J. Zerubia, A.K. Jain, eds., *Lecture Notes in Computer Science*, IEEE Computer Society, pp. 137–144 (2001).

[5] S. Frisken, R. Perry, A. Rockwood, T. Jones, Adaptively sampled distance fields: a general representation of shape for computer graphics, Computer Graphics (SIGGRAPH 00 Proceedings), pp. 249–254 (2000).

[6] M. Garland, P. Heckbert, Surface simplification using quadric error metrics. *Computer Graphics (SIGGRAPH 97 Proceedings)*, pp. 209–216 (1997).

[7] S. Gibson, Using distance maps for accurate surface representation in sampled volumes, *IEEE Symposium on Volume Visualization*, pp. 22–30 (1998).

[8] M. Kass, A. Witkin, D. Terzopoulos, 1988. Snakes: Active contour models, *Intern. J. Computer Vision* **1**, pp. 321–332 (1988).

[9] L. Kobbelt, J. Vorsatz, U. Labsik, H.-P. Seidel, A Shrink Wrapping Approach to Remeshing Polygonal Surfaces, *Computer Graphics Forum (Eurographics '99 issue)* **18**, C119–C130 (1999).

[10] L. Kobbelt, M. Botsch, U. Schwanecke, H.-P. Seidel, Feature sensitive surface extraction from volume data, *SIGGRAPH 01 Proceedings*, pp. 57–66 (2001).

[11] R. Malladi, J.A. Sethian, B.C. Vemuri, Shape modeling with front propagation: A level set approach, *IEEE Trans. Pattern Anal. and Machine Intell.* **17**, pp. 158–175 (1995).

[12] S. Osher, R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer, New York, 2003.

[13] N.M. Patrikalakis, T. Maekawa, *Shape Interrogation for Computer Aided Design and Manufacturing*, Springer, 2002.

[14] H. Pottmann, M. Hofer, Geometry of the squared distance function to curves and surfaces, in: H.-C. Hege, K. Polthier, eds., *Visualization and Mathematics III*, Springer, pp. 223-244, 2003.

[15] H. Pottmann, S. Leopoldseder, M. Hofer, Approximation with active B-spline curves and surfaces, Proc. of Pacific Graphics 2002, Beijing, *IEEE Computer Society*, pp. 8–25 (2002).

[16] H. Pottmann, S. Leopoldseder, M. Hofer, Simultaneous registration of multiple views of a 3D object, *Intl. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. XXXIV, Part 3A, Commission III, pp. 265–270 (2002).

[17] S. Rusinkiewicz, M. Levoy, Efficient variants of the ICP algorithm, *Proc. 3rd Int. Conf. on 3D Digital Imaging and Modeling*, Quebec, Springer-Verlag, 2001.

[18] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London, 1982.

[19] J.A. Sethian, *Level Set Methods and Fast Marching Methods*, Cambridge University Press, 1999.

[20] Yang, H., Wang, W., Sun, J., *Control Point Adjustment for B-Spline Curve Approximation*, submitted to CAD.

[21] H.K. Zhao, Fast Sweeping Method for Eikonal Equations I, submitted to *SIAM Numerical Analysis*.