IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 13, NO. 6, PP. 1512-1519, 2007

Efficient Surface Reconstruction using Generalized Coulomb Potentials

Andrei C. Jalba and Jos B. T. M. Roerdink *Senior Member, IEEE*

Abstract—We propose a novel, geometrically adaptive method for surface reconstruction from noisy and sparse point clouds, without orientation information. The method employs a fast convection algorithm to attract the evolving surface towards the data points. The force field in which the surface is convected is based on generalized Coulomb potentials evaluated on an adaptive grid (*i.e.*, an octree) using a fast, hierarchical algorithm. Formulating reconstruction as a convection problem in a velocity field generated by Coulomb potentials offers a number of advantages. Unlike methods which compute the distance from the data set to the implicit surface, which are sensitive to noise due to the very reliance on the distance transform, our method is highly resilient to shot noise since global, generalized Coulomb potentials can be used to disregard the presence of outliers due to noise. Coulomb potentials represent long-range interactions that consider all data points at once, and thus they convey global information which is crucial in the fitting process. Both the spatial and temporal complexities of our spatially-adaptive method are proportional to the size of the reconstructed object, which makes our method compare favorably with respect to previous approaches in terms of speed and flexibility. Experiments with sparse as well as noisy data sets show that the method is capable of delivering crisp and detailed yet smooth surfaces.

Index Terms—Surface reconstruction, Implicit surfaces, Octrees, Generalized Coulomb potentials, Polygonization.

1 Introduction

Surface reconstruction is challenging because the topology of the real surface is unknown, acquired data can be non-uniform and contaminated by noise, and reconstructing surfaces from large data sets can be prohibitively expensive in terms of computations or memory usage. A lack of information about the surface orientation at the acquired samples may further complicate the problem.

We propose a novel, geometrically adaptive method for surface reconstruction without using orientation information. It employs a fast convection algorithm (inspired by the tagging method of Zhao *et al.* [31]) to attract the evolving surface (*i.e.*, the current approximation to the final surface) towards the data points. The force field in which the surface is convected is based on generalized Coulomb potentials evaluated on an adaptive grid (*i.e.*, an octree) using the fast, hierarchical algorithm of Barnes and Hut [6]. The potentials are used to attract the evolving surface towards the data points and to remove outliers due to noise. When the convection process ends, the characteristic function χ (defined as 1 at points inside the object, and 0 at points outside) yields the final implicit surface, which is polygonized using a meshing algorithm based on tetrahedral subdivision of octree cells [27].

Formulating surface reconstruction as a convection problem offers a number of advantages. Most methods for implicit surface fitting compute a signed distance function from the given point data and represent the reconstructed surface as an iso-contour of this function. However, these approaches are very sensitive to outliers (shot noise) due to the very reliance on the distance transform. By contrast, global generalized Coulomb potentials can be used to disregard the presence of outliers due to noise. Some methods first locally fit the data and then combine these approximations by blending the locally fitting (basis) functions. Unlike this, Coulomb potentials represent long-range interactions for all data points at once, and thus convey global information. As computing Coulomb potentials is of paramount importance in a broad variety of problems, we can rely on a very efficient method for computing Coulomb potentials, to arrive at reconstruction with scalable computational time and memory requirements.

Both authors are with the Institute for Mathematics and Computing Science, University of Groningen, The Netherlands.
E-mail: andrei@cs.rug.nl, j.b.t.m.roerdink@rug.nl.

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

The main contributions of this paper include:

- An improved geometrically-adaptive method for surface reconstruction based on an implicit surface representation which only requires information about the position of the samples and is robust to the presence of noise and missing data.
- A hierarchical and adaptive representation based on octrees, which allows scalable reconstruction at different levels of detail.
- New algorithms for surface reconstruction based on convection of surfaces in generalized Coulomb potentials.

2 Related work

Popular explicit surface representations are based on parametric (*e.g.*, NURBS, B-spline and Bézier patches) and triangulated surfaces. Major drawbacks of parametric representations are that patches should be combined to form closed surfaces, which can be very difficult for arbitrary data sets, and noisy data sets are difficult to deal with. Triangulated surfaces are usually based on tools from computational geometry, such as Delaunay triangulations and their duals, Voronoi diagrams. Most methods in this class extract subsets of faces of Delaunay triangulations to yield the reconstructed surface [3, 4, 15]. They exactly interpolate the data, and therefore, are rather sensitive to noise. Moreover, inserting hundreds of thousands of points into a triangulation is computationally expensive. Examples are Alpha Shapes [15], the (Power) Crust algorithm [3,4], and the Ball-Pivoting algorithm [7]. Recently, Giesen and John [16] introduced the notion of flow in computational geometry, and Scheidegger *et al.* [28] proposed an adaptive method based on the Moving Least-Squares algorithm.

Much research is devoted to efficient reconstruction methods relying on implicit surface representations, as these offer a number of advantages. Compared to explicit methods, implicit ones elegantly deal with objects of arbitrary topology, blend and perform Boolean operations on surface primitives, and fill holes automatically. The traditional approach is to compute a signed distance function and represent the reconstructed implicit surface by an iso-contour of this function [5,8,12,17]. This requires a way to distinguish between the inside and outside of closed surfaces. E.g., Hoppe *et al.* [17] approximate the normal at each data point by fitting a tangent plane in its neighborhood, using principal component analysis. Zhao *et al.* [31] use the level-set formalism [26] for noise-free surface reconstruction. Their method handles complicated topology and deformations, and the reconstructed surface is smoother than piecewise linear. The main drawback is the sensitivity to shot noise, due to reliance on the distance

transform. The work of Zhao *et al.* [31] inspired researchers from the computational-geometry community to develop 'geometric convection' algorithms [2, 11] in the context of surface reconstruction. These methods deform a closed oriented pseudo-surface embedded in the 3D Delaunay triangulation of the sampled points, and yield the reconstructed surface as a set of oriented facets of the triangulation. Although these methods tolerate small amounts of Gaussian noise, they are computationally expensive.

More recently, blending globally or locally supported implicit primitives (*e.g.* Radial Basis Functions or RBFs) became the favorite technique [10, 13, 21, 23]. Turk and O'Brien [30] and Carr *et al.* [10] use globally supported RBFs by solving a large and dense linear system of equations. Since the ideal RBFs are globally-supported and non-decaying, the solution matrix is dense and ill-conditioned [20]. Moreover, these methods are very sensitive to noise because local changes of the positions of the input points have global effects on the reconstructed surface. Morse *et al.* [23] and Ohtake *et al.* [25] use compactly-supported RBFs to achieve local control and reduce computational costs by solving a sparse linear system. Dinh *et al.* [13] use RBFs and volumetric regularization to handle noisy and sparse range data sets. Recently, Ohtake *et al.* [24] proposed the so-called 'partition of the unity implicits', which can be regarded as the combination of algebraic patches and RBFs. Carr *et al.* [9] further address surface reconstruction from noisy range data by fitting a RBF to the data and convolving with a smoothing kernel during the evaluation of the RBF. Kojekine *et al.* [21] use compactly-supported RBFs and an octree data structure such that the resulting matrix of the system is band-diagonal, thus reducing computational costs. More recently, Kazhdan [19] proposed to use the Fourier transform for computing the characteristic function of the solid model, and then standard iso-surfacing techniques (*i.e.*, marching cubes) to triangulate its boundary. Although (small) displacements of the positions and normals of the samples are handled well, the temporal and spatial complexities are cubic in the (uniform) grid resolution. An improved, geometrically adaptive method [20] with quadratic complexities, formulates reconstruction as a Poisson problem which is then tackled using efficient Poisson solvers. However, both methods assume that orientation information is available. Kolluri *et al.* [22] introduce a noise-resistant algorithm for watertight reconstruction based on spectral graph partitioning. Since the local spectral partitioner uses a global view of the model, their algorithm can ignore outliers and patch holes in under-sampled regions. The method is computationally expensive as it solves an eigenvalue problem on a large graph formed by a subset of the Voronoi vertices of the input set. Tang and Medioni [29] use tensor-voting to develop a method which is resilient to noise and copes well with sparse data. The downside of this method is its cubic spatio-temporal complexities. In [18] regularized membrane potentials are used to aggregate the input points on a uniform grid, then a labeling algorithm similar to that in [31] is used to define an implicit surface, which is smoothed using a mass-spring system and as a final step polygonized. However, this method has cubic complexities similar to the FFT method in [19].

Similarly to RBF approaches and Poisson reconstruction, our method creates smooth surfaces that approximate noisy data, and combines some positive aspects of both global and local approximation schemes. Convection is performed in a global field (Coulomb force field) which does not rely on computing local neighborhoods for blending. However, *after* the indicator function of the solid has been computed, we do employ blending based on compactly-supported kernels to produce smooth triangulated surfaces.

3 The Proposed Method

Let S denote an input data set of samples (points, lines, etc.) lying on or near the surface $\partial \boldsymbol{M}$ of an unknown object M. The problem is to accurately reconstruct the indicator function χ of M, and then to approximate the surface $\partial \boldsymbol{M}$ by a watertight, smooth triangulated iso-surface.

Given a flexible, arbitrary enclosing surface $\boldsymbol{\Gamma}$, the reconstruction problem is formulated as the convection of $\boldsymbol{\Gamma}$ in a conservative velocity field $\boldsymbol{v} = -\nabla\phi$ created by the input data points, as described by the differential equation

$$\frac{d\boldsymbol{\Gamma}}{dt} = \boldsymbol{v}\left(\boldsymbol{\Gamma}(t)\right) = -\nabla\phi\left(\boldsymbol{\Gamma}(t)\right). \tag{1}$$

We assume that ϕ is the *electric scalar potential* (Coulomb potential), such that the velocity field becomes the electric field $\boldsymbol{v} = \boldsymbol{E} \equiv -\nabla\phi$. We regard the sample points s_i, $s_i \in S$, as electric charges q_i with position vectors $\boldsymbol{r}_i$, $i = 1,2,\ldots N$ generating a charge distribution ρ. Hence we can solve for ϕ by numerically approximating Poisson's equation

$$\nabla^2\phi = -\frac{\rho}{\varepsilon_0}. \tag{2}$$

Solving for Poisson's equation in this case has the problem that the long-range nature of Coulomb interactions is not properly taken into account. Moreover, Poisson's equation requires a continuous charge distribution and not a set of discrete charges, which are in fact just a set of singularities. This problem has been addressed in [20] by convolving the input points with a Gaussian kernel prior to solving Poisson's equation. Additionally, in [20] the authors *directly* solve a Poisson equation for the characteristic function, assuming however that the orientation of the sample points is available. Discretizing the superposition integral, the potential ϕ is a sum of potentials generated by each charge taken in isolation. However, this potential is inadequate for our purposes (see Fig. 2). Therefore, we use higher order *generalized* Coulomb potentials which decay faster with distance than as inverse of distance, *i.e.*,

$$\phi(\boldsymbol{r}_i) = \sum_{j \neq i} \frac{q_j}{|\boldsymbol{r}_i - \boldsymbol{r}_j|^m}, \tag{3}$$

where $m > 1$ and we removed the constants appearing in the physical formulation. Note that similar potentials have been successfully used for computing medial axes of 3D shapes [1]. In our case a high-order potential should be used for noise-free data when an accurate reconstruction is desired. By contrast, we show that for noisy data a smaller-order potential can be used to detect and remove outlier data.

During convection generalized Coulomb potentials have to be evaluated not only at the sample positions, but at all centers of the (adaptive) grid cells. This is required by the fast convection algorithm (described in Section 4.3) which, starting from the bounding box of the grid, follows increasing paths of the scalar field until regional maxima (corresponding to the sample points) and ridges are reached. As it sweeps the volume, this algorithm labels grid cells as exterior, so that after propagation the remaining cells qualify as interior to the surface. Naive evaluations of these potentials at *all* grid positions is expensive for large grid resolutions. So we need to use fast adaptive solvers (see Section 4.2) to approximate them.

Having labeled the cells of the adaptive grid (octree) as exterior and interior to the surface, and thus equivalently determined χ, we compute a smoothed version $\tilde{\chi}$ of χ as the weighted sum of contributions of nearby cells (estimated during polygonization). The weights are obtained by evaluating the quadratic (approximating) C^1–continuous B-spline of Dodgson [14]. This has second order interpolation error, and since it evaluates to zero and has vanishing derivatives at the boundary it is conducive to stability. Moreover, it is inexpensive to compute since it requires only three multiplications and three additions for each evaluation.

4 Implementation

4.1 Adaptive octree-based approximation

Discretizing the problem on a regular 3D grid is impractical since the memory for maintaining such a uniform structure becomes prohibitive for fine-grain reconstruction. However, since accurate representations are only required close to the data set, we can use an adaptive grid based on an octree to efficiently evaluate Coulomb potentials and to represent and triangulate the implicit function.

Given samples $s_i \in S$, $i = 1,2,\ldots,N$ regarded as particles with position $s_i.p$ and charge $s_i.q$, and a maximum tree depth D, the octree O is

built by calling the function **octInsert** shown in Algorithm 1 for each particle, with n set to the root node r. When the maximum tree depth D is reached for a non-empty leaf l, $l \in O$, the node is not further subdivided if a new particle is to be assigned to it. Instead, the centroid and total charge of the subset Z of all particles s_k that would have been assigned to the subtree rooted at l in an infinitely-deep octree are computed and stored in a new particle s_s that is assigned to l. All particles s_k can now be discarded, as they are no longer needed. Effectively we implement a low-pass filter acting on particle positions, since the given (maximum) resolution ($1/2^D$) cannot be exceeded anyway, *i.e.*,

$$s_s.p \leftarrow \frac{\sum_{s_k \in Z} s_k.q \; s_k.p}{\sum_{s_k \in Z} s_k.q}, \quad s_s.q \leftarrow \sum_{s_k \in Z} s_k.q \tag{4}$$

When initially all particles have the same charge (we use $q = 1$), the total charge of such particles will be larger than that of usual particles assigned to non-empty leaves.

Algorithm 1 Insert particle s_i in the subtree rooted at n.

function **octInsert**(s_i, n)
  if subtree rooted at n contains more than one particle **then**
    determine child c of n which contains particle s_i;
    octInsert(s_i, c);
  else if subtree rooted at n contains exactly one particle **then**
    if $n.depth \geq D$ **then**
      determine particle s_j contained in n;
      create new particle s_s with charge $s_s.q = s_i.q + s_j.q$
        and position $s_s.p = (s_i.q \cdot s_i.p + s_j.q \cdot s_j.p)/s_s.q$;
      assign s_s to n and discard particles s_i and s_j;
    else
      add n's eight children to the octree;
      move particle s_j already in n into the child in which it lies;
      let c be the child in which s_i lies;
      octInsert(s_i, c);
  else
    store s_i at node n;

After octree construction the octree is balanced (by subdividing nodes until any two neighboring cells differ at most by one in depth), as required by both the meshing (see Section 4.4) and convection algorithms (see Section 4.3 and Fig. 2).

4.2 Fast evaluation of the Coulomb potentials

To evaluate the generalized Coulomb potentials within any desired precision on the adaptive grid obtained by stacking the cells of the balanced octree, we rely on the fast, hierarchical algorithm of Barnes and Hut [6], thus trading accuracy for speed. The Barnes-Hut algorithm can be summarized as follows: (i) construct an octree where each leaf contains at most one particle, (ii) for each octree cell, compute the centroid and total charge for the particles it contains, (iii) traverse the tree to evaluate the potential at any desired grid location. The first step is accomplished using the function **octInsert** in Algorithm 1. The second step is implemented in a depth-first-search traversal of the balanced tree O_b, in which the total charge $n.q$ and the centroid $n.c$ of each node $n \in O_b$ are computed (using Eq. (4)) and stored at each node.

Then we come to the core of the Barnes-Hut algorithm, evaluating the potential at an arbitrary location $\boldsymbol{p}$. A small positive test charge q_t (with value $q_t = 1$) is placed at position $\boldsymbol{p}$, *i.e.*,

$$\phi(\boldsymbol{p}) = q_t \sum_{i=1}^{N} \frac{q_i}{|\boldsymbol{r}_i - \boldsymbol{p}|^m}. \tag{5}$$

Then one computes the ratio between the size L of a given cell and the distance r_i, from $\boldsymbol{p}$ to the centroid of the cell, see Fig. 1. If the ratio is smaller than θ (a user-supplied accuracy parameter), the potential is computed from the total charge and centroid of the particles in the cell. Otherwise, computation continues with the children of the node associated to the current cell, see Algorithm 2.

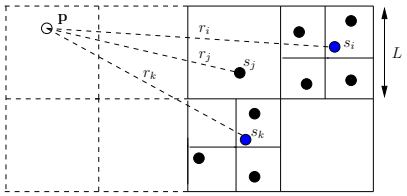

Fig. 1. Estimation of the Coulomb potential at an arbitrary grid position $\boldsymbol{p}$. If the ratio $L/r_i < \theta$ the direct contribution is replaced by one contribution due to all particles in the box. Grid cells are only subdivided during the octree construction, see Algorithm 1.

Algorithm 2 Evaluate the potential at $\boldsymbol{p}$ due to all particles in the cell associated with octree node n.

function **computePotential**($\boldsymbol{p}$, n) returns pot
  $pot = 0$;
  if n contains one particle s_i **then**
    $pot =$ compute direct potential due to s_i;
  else
    $r =$ distance from $\boldsymbol{p}$ to centroid $n.c$;
    $L =$ size of the cell associated with n;
    if $L/r < \theta$ **then**
      compute pot due to $(n.q,\ n.c)$;
    else
      for all eight children m_i of n **do**
        $pot = pot +$ **computePotential**($\boldsymbol{p}$, m_i);
  return pot;

4.3 Fast convection based on tagging

We revise the tagging algorithm of Zhao *et al.* [31] to find the exterior and interior octree cells to the surface, *i.e.*, to compute the characteristic function χ of the model on octree grids.

Given the balanced octree O_b, the task is to classify its leaf cells as either exterior, interior or boundary at each depth of the octree, starting at the coarsest grid resolution and moving gradually towards the highest resolution at tree depth D. This is done in a breadth-first-search tree traversal based on a queue Q. Initially, all octree cells at all depths are labeled as interior, and the root of the tree as well as a *sentinel* node s are added to Q. Then, the octree is traversed by removing one node at a time from Q. Like Zhao *et al.* we also use a sorted heap H storing leaf cells at the current depth d, but the cells are sorted in *increasing* order of their keys (potential values at the cell centers). If the current node n taken out from Q is the sentinel node s, this signifies that all *accessible* leaves at the current depth d, $d = 1, 2, \ldots, D$ have been collected in the heap H (see below), and the marching of the exterior boundary at level d towards the data points can be performed by Algorithm 3. After marching at level d ends, the current depth d is incremented, a new sentinel is added to Q, and a node (at the new depth) is extracted from Q and assigned to the current node n. The traversal continues by adding all children of n to Q. Then, whether or not the current node was the sentinel one, it is labeled as exterior if the node touches the bounding box. Otherwise, if (i) it is an interior leaf node, and (ii) has at least one exterior neighbor at a depth smaller or equal to its own, and (iii) the potential evaluated at its center is larger or equal to that evaluated at the center of its neighbor, it is labeled as trial and added to the heap H. The process is repeated by extracting a new node from Q until the whole tree has been traversed.

The computation is summarized in Algorithm 3. Compared to the tagging method there are some slight modifications. First, we replaced the maximum heap with a minimum one, as we march towards local maxima and ridges of generalized Coulomb potentials. A more subtle modification was required due to the fact that during marching, even if the octree is balanced, some interior neighbors of a node may lie at a depth smaller (by one) than its own. Since these nodes will never be reached and thus inserted in the heap, they have to be resolved immediately. This is handled by subdividing any such neigh-

Algorithm 3 Label cells as exterior or boundary.

```
function marchExterior(H)
  while not empty heap H do
    extract node n from H;
    p_n = computePotential(n.center, root);
    for each leaf m, neighbor of n, m.depth ≤ n.depth do
      if m is interior and m.depth < n.depth then
        subdivide m;
        find neighbor k of n among the children of m;
        assign m = k and continue;
      if m is interior then
        p_m = computePotential(m.center, root);
        if p_m + ε < p_n then
          label n as boundary;
          continue with extracting a new node from H;
    label n as exterior;
    for each leaf m, neighbor of n do
      if m is interior then
        label m as trial;
        insert m onto H;
```

bor and continuing the marching process normally. Additionally, a user-specified parameter ε has been introduced to allow the marching process to continue if the potential at a neighboring node is not small enough compared to that of the current node. For noise-free data sets we set this parameter to $\varepsilon = 0$, whereas for data sets corrupted by outliers we allow for some variation and set ε to a value larger than zero (see Section 5.5).

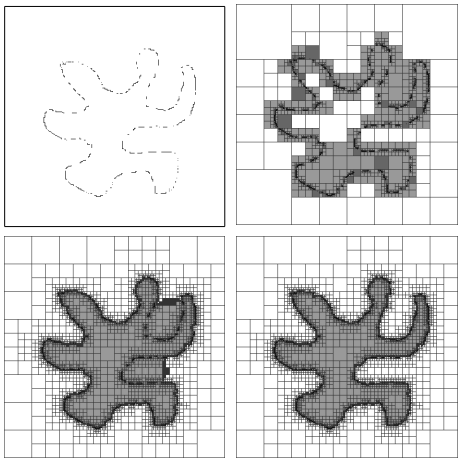

Fig. 2. Example of 2D curve reconstruction. *Top-left*: input set; *Top-right*: labeling fails if the octree is unbalanced (highly non-uniform data set); *Bottom-left*: Labeling can still fail for slower-decaying potentials ($m = 1$ in Eq. (3)) even on a balanced tree; *Bottom-right*: correct result on a balanced tree with $m = 5$.

4.4 Smoothing and polygonization

We define the implicit surface as the zero level set of the (characteristic) function χ (on the octree) constructed as follows. In a depth-first-search traversal, the total charge associated with boundary and non-leaf nodes is set to zero, whereas that of exterior nodes is set to one and that of interior ones is set to minus one. During polygonization, a smoothed version $\tilde{\chi}$ of χ is computed by the weighted sum of contributions of nearby cells, where the weights are obtained by evaluating the B-spline kernel mentioned at the end of section 3, *i.e.*,

$$\tilde{\chi}(\boldsymbol{r}) \equiv q(\boldsymbol{r}) \, \frac{\sum_{n \in O_b} n.q \, W_n(\boldsymbol{r})}{\sum_{n \in O_b} W_n(\boldsymbol{r})}, \; W_n(\boldsymbol{r}) = W\left(\frac{3|n.\boldsymbol{center} - \boldsymbol{r}|}{2h_n}\right).$$

Here $q(\boldsymbol{r}) = 1$ is a test charge at location $\boldsymbol{r}$, and $n.q$, h_n and $n.\boldsymbol{center}$ are the total charge, kernel support and center of the grid cell associated with n, respectively. The support h_n is set to $h_n = s \cdot G/2^{n.depth}$, where s is a user-supplied parameter controlling the amount of smoothing; $n.depth$ is the octree depth of node n, and G is the size of the computational domain. With some abuse of terminology, we call $\tilde{\chi}$ the smoothed characteristic function, although it takes values in the interval $[-1, 1]$.

Algorithm 4 Evaluate $\tilde{\chi}$ at location $\boldsymbol{r}$.

```
function computeChi(r, s) returns F
  n = locate leaf containing r;
  h_n = G/2^{n.depth};
  f = n.q W_n(r); sf = W_n(r);
  label n as trial and add n to Q;
  while not empty queue Q do
    dequeue node n from Q;
    for each neighbor m of n do
      if m is not trial then
        if m has children then
          m = unlabeled child of m closest to r;
        h_m = G/2^{m.depth};
        d_m = r.dist(m.center);
        if d_m ≤ h_m and d_m ≤ 2·s·h_n then
          f = f + m.q W_m(r); sf = sf + W_m(r);
          label m as trial and add it to Q;
  label all visited nodes as not-trial;
  return (F = f/sf);
```

We limit the number of considered cells surrounding the cell associated to node n and which contains $\boldsymbol{r}$, such that only cells in the neighborhood $\mathcal{N}_{O_b}(n)$ of n, with $\mathcal{N}_{O_b}(n) = \{m \mid \boldsymbol{r}.dist(m.\boldsymbol{center}) \leq 2 \cdot s \cdot h_n\}$ are visited. Therefore, we designed a fast flood-fill algorithm based on a queue Q which implements blending as just described, the pseudo-code of which is shown in Algorithm 4.

To extract the iso-surface, we use a method based on tetrahedral subdivision of octree leaves [27]. In a depth-first-search traversal of the octree, all grid cells corresponding to boundary nodes are collected and their faces triangulated. We walk around each face in a fixed direction, *e.g.*, counterclockwise as seen from the positive axis perpendicular to that face, starting at the corner vertex with the largest coordinates. By checking whether the neighboring cells are subdivided, this results in a unique sign pattern which helps us to triangulate the face. Once all its faces have been triangulated, the cell is tetrahedrized connecting each of its triangles to the center of the cubic cell. When a new tetrahedron is to be triangulated, we check whether the implicit function intersects it. The final, smoothed implicit surface is defined as the iso-surface at level zero of the function $\tilde{\chi}$. To test for intersections, we estimate the value of $\tilde{\chi}$ at the vertices of each cubic cell and at the cell center using Algorithm 4. Only if the surface intersects the tetrahedron it is triangulated as described above. Note that the resulting triangulated surface is guaranteed to be manifold by the construction of $\tilde{\chi}$.

To speed up the computations, the values of $\tilde{\chi}$ are cached using a hash map, whose keys are obtained by hashing 3D positions of cell vertices. In this way memory usage is kept low, since only an integer (the final hashing key) and a floating point number (the function value) have to be stored for each vertex of the *visited* cells. The number of triangles generated by this method will be larger than that of the marching cubes algorithm (see [20, 27]), but there are no ambiguous polygonization cases, the algorithm is straightforward and results in better interpolation of mesh vertices.

The method is geometrically adaptive in the sense that if the samples are non-uniformly distributed, during polygonization, larger and fewer triangles are generated in regions of small sampling density, whereas smaller and more triangles will be generated in regions with high sampling density. However, as the resolution of the adaptive octree grid reflects the distribution of the sample points, if the input data

set is uniform, then the final mesh will have many triangles of similar sizes.

4.5 Cost analysis

Let N denote the number of samples in the data set S. Assuming that the maximum octree depth D is such that to each leaf a sample is assigned (*i.e.*, the grid resolution agrees with that of the samples), then the number of leaves is approximately equal to the number of samples, *i.e.*, $2^{3D} \approx N$. Then the total number M of octree nodes is $M \equiv \sum_{i=0}^{D} N/2^{3i} \approx N \approx 2^{3D}$. Since balancing does not increase the order of complexity, constructing the balanced octree O_b is done in $O(M \cdot \log M)$.

Since each interior cell is visited at most once during convection, the complexity of the marching procedure is $O(M \cdot \log M)$, where $\log M$ comes from the heap sort algorithm. Since the complexity of evaluating the Coulomb potential (see Algorithm 2) at the center of a grid cell is $O(\log M)$, the total complexity of the fast convection algorithm is $O(M \cdot \log^2 M)$. Further, considering that a constant number of cells are visited for evaluating $\tilde{\chi}$ at a point (see Algorithm 4), the complexity of the polygonization step is linear in the total number of nodes. Accordingly, in the worst case, if the tree depth is increased by one level, the temporal complexities of both the convection and polygonization steps increase by a factor of eight. The same worst-case total complexity is reached when the resolution of a uniform grid is doubled. However, when using an adaptive octree grid this does not happen since the octree is not complete, *i.e.*, the samples are non-uniformly distributed on the grid (see Section 5.1). Note that our method has similar spatial worst-case complexities.

5 Results

We present several results obtained using the proposed method, in a large variety of settings, using a system with two dual Opteron processors and 8 GB of RAM. The parameters of the method were set as follows. Given a desired grid resolution (maximum octree depth D), the size of the computational domain G is set to $G = 2^D$. Parameter D controls the tradeoff between the accuracy of the reconstruction versus the computational requirements. If reconstruction is to be performed at maximum resolution, D should be set such that each leaf of the octree is assigned a sample point. The parameter θ controlling the accuracy of the computations of generalized Coulomb potentials was fixed to $\theta = 0.9$. We experimented with different settings of θ in the range of $0.5 - 0.95$, without noticing differences in the quality of the reconstructed surfaces. The factor s determining the support radius of the smoothing kernel in Algorithm 4 was set to $s = 2.0$. The larger the value of this parameter the smoother the final surface becomes, albeit at the expense of larger computational time; we experimented with values in the range of $1.6 - 2.5$. The settings of the remaining parameters (m and ε) will be mentioned in the text. On the setting of parameter m we mention that similar generalized Coulomb potentials have been successfully used for computing medial axes of 3D shapes [1], and it was shown that as the potential order becomes arbitrarily large, the axes approach those computed using the shortest distance to the border [1]. Since in general it is possible to revert the medial axis transform to obtain an approximation of the input shape, this suggests that as the order of the potential is increased, a more accurate approximation can be obtained, which in the limit yields the original shape. In our case this means that for noise-free data, when an accurate reconstruction is desired, a high-order potential should be used.

5.1 Multi-resolution

The method delivers representations at different scales, where the maximum depth of the tree D plays the role of the scale parameter. Fig. 3 shows reconstruction results of a dragon model ($3,609,600$ samples) at different octree depths. As the depth of the octree is increased by one, details at higher resolutions are captured, while the computational time and number of triangles increase roughly by a factor of four, whereas the memory overhead increases by a factor of two. Some statistics are given in Table 1.

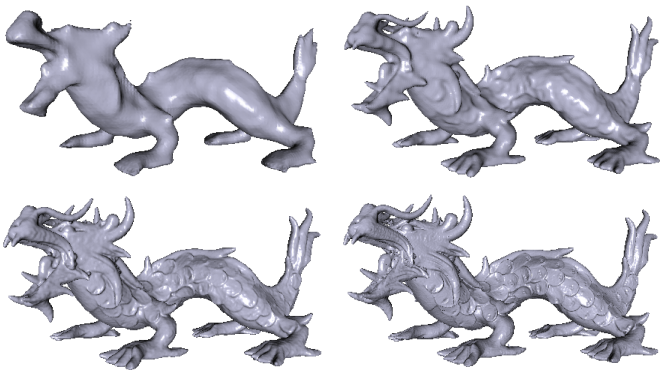

Fig. 3. Example of reconstruction at various octree depths: *Top-left*: $D = 7$, *top-right*: $D = 8$, *bottom-left*: $D = 9$, *bottom-right*: $D = 10$.

Octree depth	Time	Peak memory	Triangles
7	3	120	98,217
8	9	146	445,912
9	37	250	1,923,982
10	186	587	8,042,874

Table 1. Computational time (seconds), peak-memory usage (megabytes) and number of triangles of the reconstructed surface as functions of the tree depth, for the large dragon model (see Fig. 3).

5.2 Comparison to other methods

We compare our results to those obtained using Power Crust [4], Multi-level Partition of Unity implicits (MPU) [24], the method by Hoppe *et al.* [17], the FFT method in [19] and the Poisson-based method [20]. The experiment was performed using the Stanford bunny data set consisting of $362,000$ samples assembled from range data. The normal at each sample was estimated as in ref. [20] when required, *i.e.*, for the MPU, FFT and Poisson methods.

The results are shown in Fig. 4. Since this data set is contaminated by noise, interpolating methods such as the Power Crust generate very noisy surfaces with holes due to the non-uniformity of the samples. The method by Hoppe *et al.* [17] generates a smooth surface, although some holes are still visible due to the non-uniform distribution of samples, which the method cannot properly handle. The MPU method yields a smooth surface without holes, but with some artefacts due to the local nature of the fitting, which does not cope well with the noise and non-uniformity of the data. Global methods such as the FFT, Poisson and our method accurately reconstruct the surface of the bunny. Table 2 provides some statistics about the methods as well as about the quality of the reconstructed surfaces. The last column shows the approximation error, which is computed as the average distance from the data points to the centroids of the mesh triangles and represents an upper bound for the average distance from the data points to the reconstructed surface. The smallest reconstruction error was achieved by the Power Crust, MPU and our method. Although the Power Crust method should have produced an interpolating surface, thus achieving a smaller upper-bound error, this does not happen in this case, as the

Method	Time	Peak memory	Triangles	Error
Power Crust	504	2601	1,610,433	4×10^{-4}
Hoppe *et al.*	82	230	630,345	6×10^{-4}
MPU	78	421	2,121,041	4×10^{-4}
FFT method	93	1700	1,458,356	6×10^{-4}
Poisson method	188	283	783,127	5×10^{-4}
Our method	79	197	2,647,986	4×10^{-4}

Table 2. Computational time (seconds), peak-memory usage (megabytes), number of triangles and reconstruction error for the Stanford bunny by different methods (see also Fig. 4).

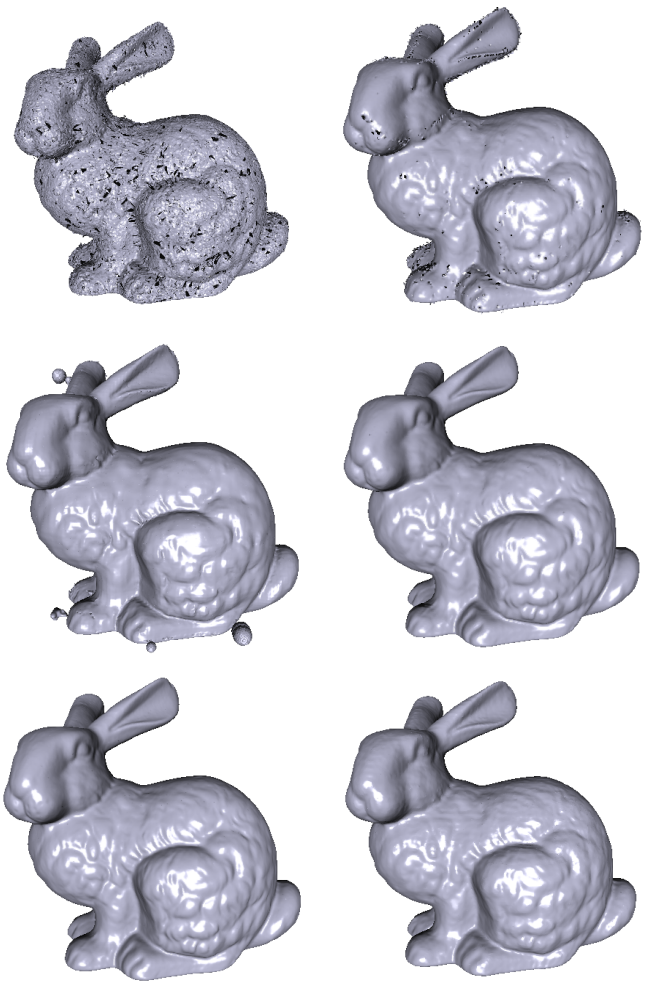

Fig. 4. Reconstruction of the Stanford bunny; *Left-to-right*, *top-to-bottom*: Power Crust, Hoppe *et al.*, MPU, FFT-based, Poisson reconstruction, and our method.

reconstructed surface contains holes. Although the surface generated by the Poisson method is slightly smoother than that of our method, the reconstruction error is higher.

The computation time of our method is comparable to that of the MPU method, which is one of the fastest geometrically-adaptive reconstruction methods according to [20, 24], while the peak memory usage of our method (at octree depth $D = 9$ with $m = 5$) remains well below those of the other methods. Although the FFT method is fast in this case, it becomes impractical at higher grid resolutions due to its cubic complexities [19]. The Poisson method is roughly two times slower than our method and has higher memory usage. It also does not cope well with noisy inputs, see Section 5.5. Finally, as our polygonizer is based on tetrahedral decomposition, the number of triangles of the reconstructed surface is quite high. However, methods for surface decimation can be used to improve this.

5.3 Range data sets

We performed an experiment on range data using three popular data sets: the 'Happy Buddha' model (2,468,000 samples), the 'dragon' model (2,198,655 samples), and the 'Armadillo' model (2,185,867 points). The results are shown in Fig. 5 and Table 3. The maximum octree depth was set to $D = 10$. When evaluating generalized Coulomb potentials for the Buddha model at a location $\boldsymbol{p}$ we only considered those cells which were in a ball of radius $r = 100$ centered at $\boldsymbol{p}$. This was necessary because, since there are no samples between the feet, our method would have connected both feet just above the pedestal of the statue. Note that this does happen with the Poisson method, see [20].

Fig. 6. Reconstruction of two large models ('Thai statue' and 'Lucy', 5 and 14 million samples, respectively) at octree level $D = 11$.

Model	Time	Peak memory	Triangles	Error
Buddha	437	860	12,804,370	6×10^{-4}
Dragon	318	586	12,034,739	5×10^{-4}
Armadillo	293	530	9,448,478	3×10^{-4}

Table 3. Computational time (seconds), peak-memory usage (megabytes) and number of triangles of the reconstructed surfaces of three range data sets (see Fig. 5).

5.4 Reconstruction of large models. Scalability

To study scaling with higher resolutions, we performed a further experiment on two large data sets, the 'Lucy' model (14 million samples) and the 'Thai statue' (5 million samples). Fig. 6 shows the reconstructions performed at a maximum octree depth of $D = 11$. The Lucy model took 21 minutes using 2.6 GB of RAM, generating a mesh with 39 million triangles. Similarly, reconstructing the Thai statue took 28 minutes using 3.0 GB of RAM (42 million triangles). Reconstructing these models with methods such as the FFT [19] or the method in [18] would require two uniform grids of size 2048^3 voxels, *i.e.*, more than 60 GB of memory.

5.5 Coping with noise

5.5.1 Shot noise

A number of outliers, expressed as a percentage of the number of input samples was added in order to produce noisy data sets. Parameters were modified as follows. First we decreased the power m in Eq. 3 to $m = 2$ (see Section 3), which produces a slower decaying field. Contributions of nearby particles are stronger, which allows marching to continue whenever outliers are encountered. However, this is not sufficient because any sample location (whether input sample or outlier) represents a singularity which is clipped and thus turned into a local maximum. Therefore we also allowed for some variation of the potentials by putting $\varepsilon = 0.1$. In this way we can detect and remove outlier data as demonstrated in Fig. 7. Note that the method still works for two times more (uniformly-distributed) outliers than data points, albeit at the cost of some visible artefacts.

Among the few methods which can tolerate shot noise is the recent one of Kolluri *et al.* [22]. However, the CPU time reported in [22] is one order of magnitude higher than ours, on the same range data set

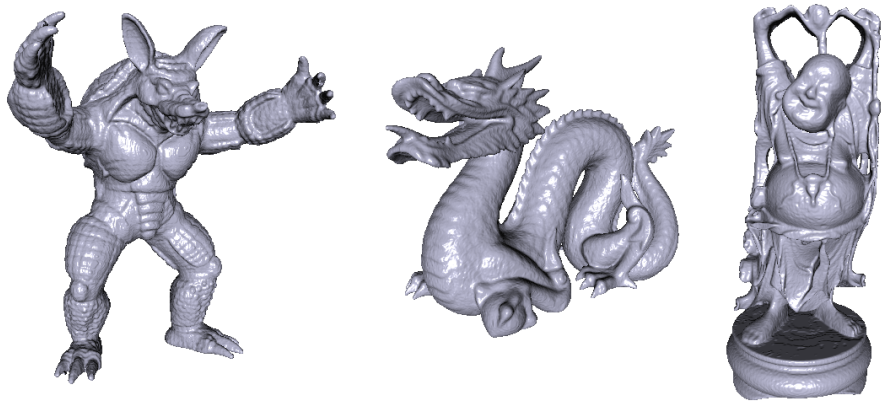

Fig. 5. Reconstruction from real-world range data sets.

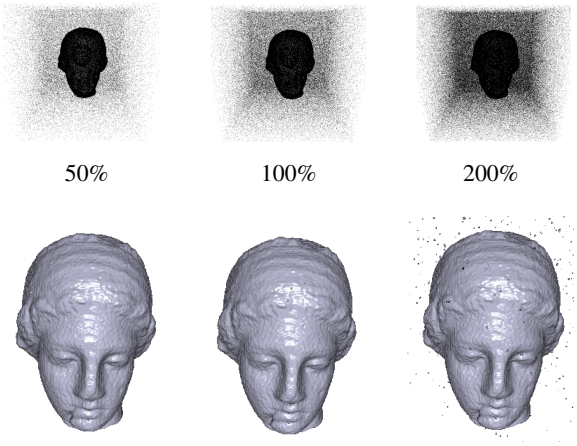

Fig. 7. Shot noise experiment. *First row*: input data sets, with the numbers representing the percentages of added noise samples; *second row*: reconstruction results (see text).

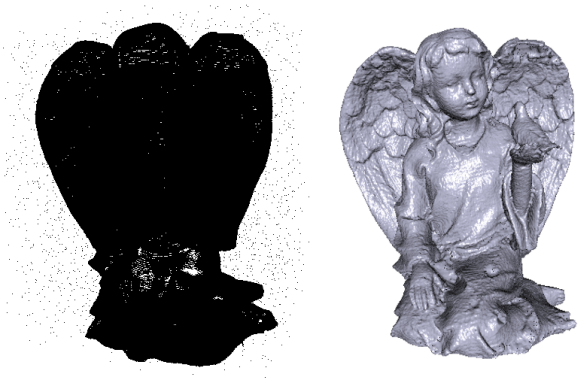

Fig. 8. Shot noise experiment. *Left*: noisy range data set with $2,008,414$ samples and $4,000$ outliers; *right*: reconstructed surface (CPU time: 168 seconds). Angel data set courtesy of the U.C. Berkeley Computer Animation and Modeling Group.

and with similar reconstruction quality (compare Fig. 1 in [22] with Fig. 8). Our reconstructed surface has genus 4, that of Kolluri *et al.* has genus 14, whilst the source object has genus 1, [22]. Although the FFT method is resilient to Gaussian noise, it does not cope well with shot noise, as this affects the *whole* Fourier spectrum and not only certain (high) frequencies. Also, it is not clear how sample normals can be computed in such cases.

5.5.2 Gaussian noise

We perturbed the positions of the input points with Gaussian noise of zero mean and different standard deviations, expressed as percentages of the length of the diagonal of the bounding box spanned by the sample points, and compared to the Poisson method [20]. The parameters of our method were set as in the previous section. Since this method requires normals at the sample points, we first reconstructed the surface (at level $D = 8$) using our method, computed per-vertex normals and then fed the Poisson method with the vertices and normals of the mesh. The results are shown in Fig. 9. Even when the coordinates of most points were randomly translated in the interval $[-8,+8]$, our method was able to reconstruct the surface, although the reconstruction is rather rough. As shown in [19], methods based on local fitting such as the MPU and some RBF methods do not tolerate well Gaussian noise, nor does the Poisson method, as shown in Fig. 9.

5.6 Limitations

Surface features smaller than the size of the smallest grid cells (at octree depth D) are not accurately reconstructed. A possible solution is to increase the grid resolution at the expense of larger computational time and memory requirements. As for any method that employs implicit representations, we assume that the surfaces to be reconstructed are closed. Fig. 10 also illustrates in 2D several failure modes of the method. If there is a deep cavity in the model, with poor sampling density along its walls, the potential across the entry to the cavity forms a ridge closing off the cavity, Fig. 10, left. Another problem appears if there are significant holes in the sampling. In this case, the ridges of the potential field do not form a closed surface and labeling fails, Fig. 10, center. Still, the method does perform intrinsic hole filling by minimal surfaces, see Fig. 10, right.

6 Conclusions

We have shown that surface reconstruction can be formulated as a convection problem of a surface in a velocity field generated by generalized Coulomb potentials, and that this offers a number of advantages. The method scales well and can be used to efficiently reconstruct surfaces from clean as well as noisy non-uniform, real-world data sets.

Further work concerns more efficient smoothing of implicit surfaces on non-uniform grids based on curvature flows and/or (anisotropic) diffusion, and improving the overall performance of the polygonization method. Of particular interest is the possibility of accelerating the computations using modern, programmable GPU hardware.

References

[1] N. Ahuja and J.-H. Chuang. Shape representation using a generalized potential field model. *IEEE Trans. Pattern Anal. Machine Intell.*, 19(2):169–176, 1997.

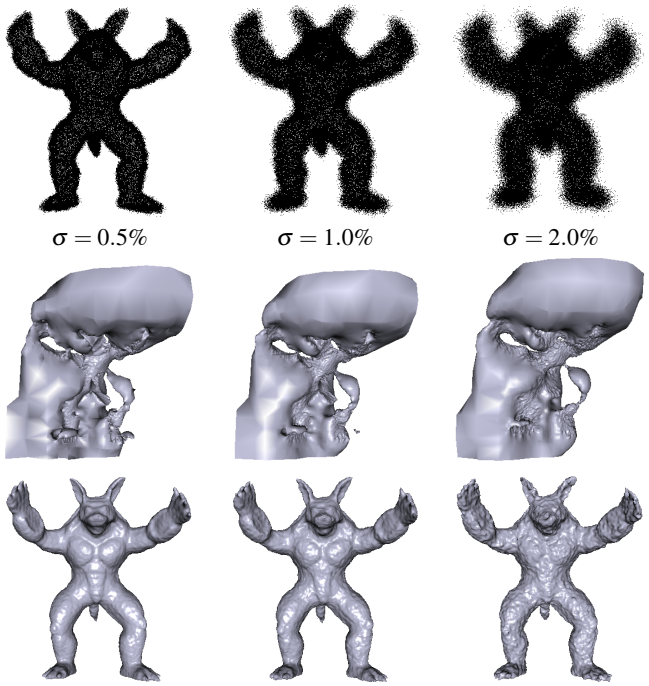

Fig. 9. Gaussian noise. *First row*: input samples and standard deviations σ as percentages of the diagonal of the bounding box; *second row*: reconstructions by the Poisson method; *third row*: reconstructions by our method.

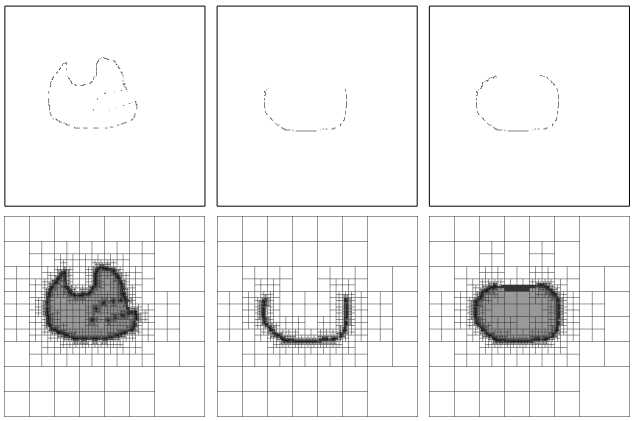

Fig. 10. Failure modes of the method. *First row*: input sets; *second row*: reconstruction results, *left*: labeling fails when there are large cavities in the object and when there are significant holes in the sampling, *center*: very large gaps cannot be bridged, *right*: smaller gaps are filled.

[2] R. Allègre, R. Chaine, and S. Akkouche. Convection-driven dynamic surface reconstruction. In *Proc. Shape Modeling International*, pages 33–42, 2005.

[3] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: Combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60(2):125–135, 1998.

[4] N. Amenta, S. Choi, and R. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19(2–3):127–153, 2001.

[5] C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. *Computer Graphics*, 29:109–118, 1995.

[6] J. E. Barnes and P. Hut. A hierarchical O(N Log N) force calculation algorithm. *Nature*, 324(4):446–449, 1986.

[7] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. on Visualization and Comp. Graphics*, 5(4):349–359, 1999.

[8] J. D. Boissonnat and F. Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *Proc. 16th ACM Symp. on Comput. Geom.*, pages 223–232, 2000.

[9] J. Carr, R. Beatson, B. McCallum, W. Fright, T. McLennan, and T. Mitchell. Smooth surface reconstruction from noisy range data. In *Proc. Graphite 2003*, pages 119–126, 2003.

[10] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proc. SIGGRAPH'01*, pages 67–76, 2001.

[11] R. Chaine. A geometric convection approach of 3-D reconstruction. In *Proc. Eurographics Symposium on Geometry Processing*, pages 218–229, 2003.

[12] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH'96*, pages 303–312, 1996.

[13] H. Q. Dinh, G. Turk, and G. Slabaugh. Reconstructing surfaces by volumetric regularization using radial basis functions. *IEEE Trans. Pattern Anal. Machine Intell.*, pages 1358–1371, 2002.

[14] N. A. Dodgson. Quadratic interpolation for image resampling. *IEEE Trans. Image Processing*, 6(9):1322–1326, 1997.

[15] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graphics*, 13(1):43–72, 1994.

[16] J. Giesen and M. John. Surface reconstruction based on a dynamical system. *Computer Graphics Forum*, 21(3):363–371, 2002.

[17] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proc. SIGGRAPH'92*, pages 71–78, 1992.

[18] A. C. Jalba and J. B. T. M. Roerdink. Surface reconstruction from noisy data using regularized membrane potentials. In *Eurographics/IEEE VGTC Symposium on Visualization*, pages 83–90, 2006.

[19] M. Kazhdan. Reconstruction of solid models from oriented point sets. In *Eurographics Symposium on Geometry Processing*, pages 73–82, 2005.

[20] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing*, pages 61–70, 2006.

[21] N. Kojekine, V. Savchenko, and I. Hagiwara. Surface reconstruction based on compactly supported radial basis functions. In *Geometric modeling: techniques, applications, systems and tools*, pages 218–231. Kluwer Academic Publishers, 2004.

[22] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Symposium on Geometry Processing*, pages 11–21. ACM Press, July 2004.

[23] B. S. Morse, T. S. Yoo, P. Rheingans, D. T. Chen, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Shape Modeling International*, pages 89–98, 2001.

[24] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H. Seidel. Multi-level partition of unity implicits. In *Proc. SIGGRAPH'03*, pages 463–470, 2003.

[25] Y. Ohtake, A. Belyaev, and H. Seidel. Multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *Proc. of Shape Modeling International*, pages 153–164, 2003.

[26] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[27] S. Plantinga and G. Vegter. Isotopic meshing of implicit surfaces. *Vis. Comput.*, 23(1):45–58, 2006.

[28] C. Scheidegger, S. Fleishman, and C. Silva. Triangulating point-set surfaces with bounded error. In *Proc. Eurographics Symposium on Geometry Processing*, pages 63–72, 2005.

[29] C. K. Tang and G. Medioni. Inference of integrated surface, curve, and junction descriptions from sparse 3-D data. *IEEE Trans. Pattern Anal. Machine Intell.*, 20:1206–1223, 1998.

[30] G. Turk and J. F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Trans. Graph.*, 21(4):855–873, 2002.

[31] H. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *Proceedings of the IEEE Workshop on Variational and Level Set Methods in Computer Vision*, pages 194–202, 2001.