Software Tools Using CSRBFs for Processing Scattered Data

Nikita Kojekine [a,*], Ichiro Hagiwara [a], Vladimir Savchenko [b]

[a] *Hagiwara's Laboratory, Department of Mechanical Sciences and Engineering, Faculty of Engineering, Tokyo Institute of Technology, 2-12-1, O-okayama, Meguro-ku, Tokyo, 152-8552, Japan.*

[b] *Faculty of Computer and Information Sciences, Hosei University, 3-7-2, Kajino-cho, Koganei-shi, Tokyo, 184-8584, Japan.*

Abstract

A set of software tools that use compactly supported radial basis functions (CSRBFs) to process scattered data is proposed in this paper. To solve problems concerning the processing of scattered data in such applications as reconstruction of functionaly defined geometric objects, surface retouching, and shape modifications, we employ a specially designed C++ software library. Thanks to the efficient octree algorithm used in this study, the resulting matrix is a band-diagonal matrix that permits handling of large data sets in a reasonable time.

The method, classes of the software library, time performance of the algorithm, and various examples of the use of the software tools are discussed.

Key words: surface reconstruction and modification, retouching, scattered data interpolation, computer graphics.

1 Introduction

Many practical surface reconstruction techniques based on measured data points require the solution of optimization problems in the fitting of surface data, as, for example, in restoration design or reverse engineering tasks. One of the applications considered in this paper, namely, reconstruction of volume geometric models, is an

* Corresponding author, tel. +81-3-5734-3556, fax. +81-3-5734-2893.
Email addresses: karlson@stu.mech.titech.ac.jp (Nikita Kojekine), hagiwara@mech.titech.ac.jp (Ichiro Hagiwara), vsavchen@k.hosei.ac.jp (Vladimir Savchenko).

attractive research direction: it makes possible to speed up the solution of reverse engineering problems by using scattered measured data, and addresses the demand for the reconstruction of volume models that allow a rich set of operations to be applied. Recently, many researchers have paid attention to possibilities opened by using radial basis functions (RBFs) for surface reconstruction of functionaly defined objects. Traditionally, constructive solid geometry (CSG) modeling uses simple geometric objects for a base model, which can be further manipulated by implementing a certain collection of operations such as set-theoretic operations, blending, or offsetting. The operations mentioned above and many others have found quite general descriptions or solutions for geometric solids represented as points (x,y,z) in space satisfying $f(x,y,z) \geq 0$ for a continuous function f. Such a representation is usually called a *function representation*. Set-theoretic solids have been successfully included in this type of representation with the application of *R*-functions and their modifications (see [1], [2]).

A vast volume of literature is devoted to the subject of scattered data reconstruction and interpolation. In most applications, Delaunay triangulation is used for 3*D* reconstruction. The main idea of Delaunay decomposition is to reconstruct a surface from non-uniform samples by connecting a subset of points that are natural neighbors in a triangulated mesh. This approach (see, for example, [3]) performs efficient Delaunay triangulation of different shape types reconstructed from their cross-sections [4].

A comprehensive overview of related studies, problems, and limitations can be found in [6], which addresses these problems and introduces a fast algorithm for constructing C^2-continuous interpolation functions. Another approach to surface reconstruction is skeletal. An implicit surface generated by point skeletons may be fitted to a set of surface points [7], but this method is rather time-consuming.

One other approach is to use methods of scattered data interpolation, based on minimum-energy properties [8], [9], [10]. These methods are widely discussed in mathematical literature (see [11], [12]). The benefits of modeling 3*D* surfaces with the help of RBFs have been recognized in [13] for Phobos reconstruction. RBFs were adopted for computer animation [14], [5] and medical applications [15], [16], and were first applied to implicit surfaces by Savchenko et al. [17]. However, the work required is proportional to the number of the grid nodes and the number of scattered data points. The amount of computation becomes significant, even for a moderate number of nodes. Special methods for reducing the processing time were developed for thin-plate splines [18]. Theoretical aspects of using radial functions in compact domains were discussed by Light in [19]. Carr et al. [20] have also recently worked on surface reconstruction of implicitly defined 3*D* objects based on the use of non-compact supported RBFs and a "both-side" approach, which has been proposed in [21]. Unfortunately, the authors of that paper did not describe their algorithm in detail. The approach taken by Turk and O'Brien in [21] of using points specified on both sides of the surface will provide successful restoration of a

surface arbitrary topology, but it involves drawbacks that lead one to suppose that it would be inefficient for applying RBFs in volume reconstruction. It is beyond the scope of this paper to discuss all these matters; let us note, however, that there is a problem of constructing or specifying off-surface points along a surface normal that also leads to a doubling of the given number of surface points. The "both-sides" approach also has a problem with surface extraction: a surface extractor can jump outside the band of non-zero points.

Methods exploiting RBFs can be divided into three groups. The first group is "naive" methods, which are restricted to small problems, but they work quite well in applications, dealing with shape transformation (see, for example, [22]). The second group is fast methods for fitting and evaluating RBFs, which allow large data sets to be modeled [23], [18]. The third and last group is compactly supported RBFs [24]. In spite of significant progress in the field of implementing RBFs and CSRBFs [25], for reconstruction purposes, it is still an open question whether it is possible to handle realistic amounts of data in real time. We suppose that RBFs and CSRBFs are suitable for moderately sized $3D$ data sets; for instance, the execution time is about 300 seconds for 36000 points, without time costs for surface extraction, as reported in [23], where the authors used commercial software for SLAE solution and employed the "both-sides" approach to reconstruct implicitly defined objects. Nevertheless, RBFs possess many features that make them very attractive for CG applications dealing with modification of geometric objects. In the simplest case, when we use sphere as a carrier solid (see the description of our surface extraction algorithm given later in section 4) to generate CSG objects we restrict our applications to the restoration of objects close to a sphere. However, using other carrier solids or dividing the point data in parts and modeling the whole object by boolean combination of the different CSRBF function, it is possible to model any type of geometry, but it can be very difficult to select appropriate carrier solid or division for the dataset.

Our software toolkit can be used both with carrier solid and "both-sides" approaches and we demonstrate some examples of restoration using both approaches later in section 4. Nevertheless, generally we avoid using "both-sides" approach because it is not sufficiently clear how to define right normal vectors for scattered data sets. Actually, we noticed in our experiments that the results of restorations heavily depend on the normal vectors.

In practice, reconstruction with RBFs consists of the following steps: sorting the data, constructing the system of linear algebraic equations (SLAE), solving the SLAE, and evaluating the functions. In fact, while the solution of the system is the limiting step, constructing the matrix and evaluating the functions to extract the isosurface may also be computationally expensive. In this paper, we attempt to solve the reconstruction problem according to the above-mentioned steps. Thus, the main goal of the ongoing project is to develop an effective library of C++ classes that can be successfully applied to computationally intensive problems of surface

reconstruction and deformation using RBF splines. The key point of the software tools we developed is the efficient octree algorithm proposed in this study: the resulting matrix is a band-diagonal matrix that permits handling of large data sets in a reasonable time.

This work is a considerably extended and improved version of a presentation we made [26]. The paper is organized as follows. In Section 2 we describe a method of surface reconstruction from a set of unorganized (scattered) points by means of radial basis functions. Our algorithms give such a reconstruction (Section 3) in two steps: sorting of data and solution of the SLAE. Section 4 gives examples of surface construction as one of the applications of our toolkit. Section 5 summarizes the results described in the paper.

2 RBF splines

For a three-dimensional arbitrary area Ω, the thin-plate interpolation is the variational solution that defines a linear operator T when the following minimum condition is used:

$$\int_{\Omega} \sum_{|\alpha|=m} m!/\alpha!(D^{\alpha} f)^2 d\Omega \to min,$$

where m is a parameter of the variational functional and α is a multi-index. It is equivalent to using the radial basis functions $\phi(r) = r^1$ or r^3 for $m = 2$ and 3, respectively, where r is the Euclidean distance between two points. Since the function $\phi(r)$ is not compactly supported, the corresponding system of linear algebraic equations (SLAE) is not sparse or bounded. Storing the lower triangle matrix requires $O(N^2)$ real numbers, and the computational complexity of a matrix factorization is $O(N^3)$. Thus, the amount of computation becomes significant, even for a moderate number of points.

Wendland in [24] constructed a new class of positive definite and compactly supported radial functions for $1D$, $3D$, and $5D$ spaces of the form

$$\phi(r) = \begin{cases} P(r),\ 0 \leqslant r \leqslant 1 \\ 0,\ r > 1 \end{cases},$$

whose radius of support is equal to 1. The function $\phi(r) = (1-r)^2$, which is an interpolated function that supports only C^0 continuity, is used. This function provides positive defined and nonsingular systems of equations. However, it may be possible to apply functions that support a higher continuity, this is a matter for further research. An investigation [27] of the smoothness of this family of polynomial basis functions shows that each member $\phi(r)$ possesses an even number of continuous derivatives.

The volume spline $f(P)$ having values h_i at N points P_i is the function

$$f(P) = \sum_{j=1}^{N} \lambda_j \phi(|P - P_j|) + p(P), \tag{1}$$

where $p = v_0 + v_1 x + v_2 y + v_3 z$ is a degree one polynomial. To solve for the weights λ_j we have to satisfy the constraints h_i by substituting the right part of equation 1, which gives

$$h_i = \sum_{j=1}^{N} \lambda_j \phi(|P_i - P_j|) + p(P_i).$$

Solving for the weights λ_j and v_0, v_1, v_2, v_3 it follows that in the most common case there is a doubly bordered matrix T, which consist of three blocks, square sub-matrices A and D of size $N \times N$ and 4×4, respectively, and B, which is not necessarily square and has the size $N \times 4$.

Heights h_i, are not necessarily values of a function defining a selected carrier solid. Arbitrary points in the Euclidean space E^n (more precisely, vectors of deviations of some defined $3D$ points) can also be used.

3 Algorithm

3.1 Sorting of scattered data

Space-recursive subdivision is an elegant and popular way of sorting scattered $3D$ data. We propose an efficient approach based on the use of variable-depth octal trees for space subdivision, which allows us to obtain the resulting matrix as a band-diagonal matrix that reduces the computational complexity.

Our first goal is to build an octal tree [28] data structure from the original point data. For each node of the tree, we need to store the following:

- a pointer to the parent node,
- 8 pointers to child nodes,
- a pointer to the list of points (empty, if this node is not a leaf).

The total amount of memory needed to store such a tree is (memory for each node) $\times$ (number of nodes) $+$ (memory needed to store points) $\times$ N. In our software implementation, we employ a standard approach for creating the tree from an initial point data set with an additional required parametric value K, which denotes the maximum number of points in the leaf. For the sake of convenience, we scale all N points in a unit cube. Afterwards we divide this cube into 8 equal sub-cubes. All

points are shared among these sub-cubes. The root of the tree corresponds to the initial cube, and the 8 sub-cubes correspond to 8 nodes linked with the root of the tree. This procedure is applied to each cube, until all K points lie in their own small cubes. If one of the 8 cubes does not contain points in the current step we should not build a sub-tree in that direction.

Afterwards we can use this tree to search for neighbors of any given point from the given N points. The neighbors are points of a sphere of radius r, whose center is located at the given point. We call this sphere an r-sphere.

To accelerate the search, the tree is simplified after creation by removing unnecessary nodes. If the node A has only one sub-node B, we can remove node the A node and replace it with B. For example, see Fig. 1 (numerals represent node numbers).

This procedure is reasonable to use only if K is small. For instance, for data in the example (a) (see Table 1) if K is set equal to 1, applying of this procedure results in removing 113 nodes from the initially created 2427 nodes.

An application-programming library was developed and contains 7 main C++ classes:

- The CDataList class defines a stack, which we use in our implementation, instead of a list,
- The CDataListItem class represents an element of the stack,
- The CGetFunctions class defines the interface for functions depending on the data storage technology,
- The CPointFunctions class inherited from CGetFunctions class contains functions connecting a COctupleTree class with the array of points, and also the condition that points in space belong to the same r-sphere,
- The CNodeNumber class serves to describe the node's number in the tree,
- The COctupleTree class defining the octal tree contains the main functions,
- The COctupleTreeNode class serves to represent the node in the octal tree.

The complexity of creation and searching using an octal tree strongly depends on the initial data and the parameter K. The depth of the tree depends on the length of the cube edge corresponding to the leaf. This length is equal to $(1/2)^M$, where M is the depth of the tree and depends on the original data. If the initial points are distributed more or less uniformly, then the tree will have sufficiently uniform filling and will be symmetric. If $K = 1$, at that time the tree will be close to a full octal tree with N leaves. The maximum complexity of searching the tree will be proportional to the depth of the tree, that is, $[log_8 N]$. A more detailed account of these algorithms, including a C++ library description, can be found in [29].

The procedure of searching for the neighbors of a point in a given r-sphere is applied several times in the application. For example, it is used in the calculation of the function 1 to sum up only the points that are neighbors of the specified point with coordinates (x, y, z). However, the first application is the construction of the

band diagonal sub-matrix $\phi(|P - P_i|)$, which accounts for a significant portion of the computational cost. Band-diagonal systems have $\alpha_1 \geq 1$ of nonzero elements to the left of the diagonal and $\alpha_2 \geq 1$ of nonzero elements to its right. In our case, these numbers are equal, because our matrix is symmetric. In our application, to store the band-diagonal matrix, we use a so-called profile form or a slightly modified Jennings envelope scheme [30]. To store the matrix A, an array can be used for diagonal elements; values of non-diagonal elements and correspondent indices of the first non-zero elements in the matrix lines are placed in two additional arrays. To make our sub-matrix band diagonal, we need to re-enumerate the initial points in a special way.

We propose the following algorithm, named "Sorting data using an octal tree" (see the pseudo-code in Fig. 2):

- Take a point from the initial data and put it in the list.
- Go through the list and for each point in the list search for the neighbors in the initial data. When new points are found, add them to the list.
- Remove from the initial array points that have been placed in the list.
- If there are no more points in the list (that is, if all points were appended in the second step), then take the first point remaining in the initial array and repeat the above steps.

As a result of this algorithm, a band with maximum size α_1, that is, the maximum number of the neighbors of a point, is obtained. The maximum complexity of this algorithm is the complexity of searching for neighbors through the octal tree for each point, that is, $N\times$ (the maximum complexity of the searching algorithm). The maximum complexity of this algorithm is the complexity of searching for neighbors through the octal tree for each point, i.e., $N\times$ (the maximum complexity of the searching algorithm). We can reduce our computational outlays by calculating the matrix and ordering of the points simultaneously. In the first step of sorting algorithm, we have the first point and a list of its neighbors. This means that we can calculate one row (and one column) of the band-diagonal sub-matrix. In the next step, we take the second point and calculate the next row of the matrix, and so on, until we have calculated the entire matrix.

After sorting we have a banded matrix as illustrated in Fig. 3; that is, the matrix has the maximum number of nonzero elements for some point. Naturally, the results of matrix construction depend on the appropriate selection of the radius of the r-sphere. Note that the special order prescribed by a sparse matrix to minimize fill-ins is not important. Note also that the half-width for the selected r cannot be decreased. Considering the following unlikely event will clarify this concept. If we connect all neighboring points, we will obtain a graph, and if this graph has a cycle, then the maximum size we will get is less than or equal to the cycle length. Thus, if the radius is quite large, then the cycle will include nearly all the points from the input data. In this case, the maximum size of the band will also be large, and we

will have an expansion of the band at some point. The position of the expansion is not important for our implementation.

3.2 SLAE solution

Note that goals in solving any sparse system are to save time and space. According to theoretical estimation and our own practical experience with the reconstruction of surfaces based on using RBFs splines, the best method of solution is the Householder method [31]. However, this method is time-consuming, which becomes a crucial drawback in practical reconstruction problems. The attractiveness of using implicit methods such as conjugate gradient methods for large sparse systems has been well recognized in various applications. If T is positive definite and symmetric, the algorithm cannot break down, in theory [32]. Conjugate gradient methods work well for matrices that are well-conditioned. In practical applications, this restriction can limit the accuracy with which a solution can be obtained, and thus we prefer to use explicit SLAE solution methods for matrices stored in profile form. The advantage of Gaussian LU decomposition [33] has been well recognized and many software routines have been developed. For a symmetric and positive definite matrix, a special factorization, called Cholesky decomposition [31], is about twice as fast as alternative methods for solving linear equations.

In our case, we have an SLAE in the form $Tx = b$, where symmetric matrix T consists of four blocks: A, B, C, and C^T. Sub-matrices A, B, and C have sizes $N \times N$, $k \times k$, and $N \times k$, respectively. A is a band-diagonal matrix, $k = 4$ for $3D$ case, $B = 0$. A combination of block Gaussian solution and Cholesky decomposition was proposed by George and Liu in [32], and in our software tools we follow their proposal.

4 Surface evaluation and extraction

The surface fitted to a set of surface data point forms a volume model of a geometric object. The general idea of our algorithm [17] is to introduce a carrier solid with a defining function f_c and to construct a volume spline $f(P)$ interpolating values of the function f_c in the points P_i. The algebraic difference between $f(P)$ and f_c describes the reconstructed solid. The algorithm consists of two steps. In the first step, we introduce a carrier solid object, which is an initial approximation of the object being searched for. In the simplest case, it can be a sphere so that $f_c(P_i) = P_{i,x}^2 + P_{i,y}^2 + P_{i,z}^2 - 1$. Then the data set r associated with the points $r_i = f_c(P_i) : i = 1, 2, \ldots, N$ is calculated at all given points. In the second step, these values are approximated by a volume spline derived for random or unorganized points.

Calculation of the resulting spline function is accelerated in our algorithm by using previously created octal tree. This also makes the rendering time very small (see Table 1). This surface can be visualized directly using an implicit ray-tracer, and can be voxelized or polygonized to extract a mesh of polygons. For the visualization of reconstructed volumes we use an implicit function modeler tool [34], [35].

Visual inspection (images in Fig. 4) allows us to judge the interpolation features of the algorithm we have discussed. We would like to note that visual inspection of different restorations shows a dependency of the accuracy of restoration on the choice of the radius of support. Actually, there is some trade-off between the efficiency of computation and the restoration quality. Visual evaluation of the result is not sufficient for an appraisal of the algorithm, especially for CAGD applications, where numerical error estimation is very important for comparison of various scattered data interpolants. To compare, some numerical measure of the error of the approximation found for a test function is needed. In practice, the root mean square measure (RMS) of the error for the test function and the maximal deviation between the reconstructed and test functions can be used. For the 2D case in [17] we have evaluated the error of approximation by RBFs for a "noisy" function. The RMS error is equal to 0.02 for the z-coordinates of 469 random points. However, we have to state that a small RMS does not guarantee correct reconstruction results. In our experience, the results of reconstruction heavily depend on the uniformity of data distribution, and in the case of reconstruction with CSRBFs they depend also on appropriate choice of the radius of support. Thus, how to evaluate the accuracy of restoration is an open question. Nevertheless, we suppose that RBFs provide sufficiently good interpolation features, and here we illustrate visually (see Fig. 5) the accuracy of reconstruction as the boolean difference between RBF and CSRBF solutions for the same dataset "head" that we have already used as an example in this section (see Fig. 4 (a) and Table 1).

Fig. 5 shows an increasing white area (difference that can be observed on the right-hand side of the reconstructed "head") caused by an inappropriate choice of the radius of support. Let us note that we have a lack of data in the lower part of the "head". Increasing the radius of support provides better reconstruction results. However, the work required for correct reconstruction of the object becomes nearly proportional to the total number of all scattered data points (see Table 2).

Note that the approach of 3D surface reconstruction taken in this study does not guarantee restoration of highly topologically complex volume objects. As noted in [36], the accuracy of the restoration strongly depends on the uniformity of the distribution of data. Moreover, RBFs demonstrate excessive blending features that lead to undesirable smoothing effects. For 3D reconstruction using cross-sectional data, in [36] it is proposed that, for m different contours in one slice, m different function descriptions of separate contours must be used and that union of the m carrier functions defines the description of the reconstructed 2D object. That is, introducing m carrier functions allows us to localize different contours situated

in one slice to avoid an excessive blending or "spilling" of contours. For the 3*D* case such an approach looks exceedingly complicated. Nevertheless, we have to notice that the results of reconstruction do not depend on the choice of the carrier functions. Naturally, the approach taken by Turk and O'Brien in [21] of using points specified on both sides of the surface can be applied (see Fig. 6 (c)).

5 Conclusions

In this paper, we thoroughly investigated the problem of development C++ tools based on the use of CSRBFs splines for implementation in various CG applications. The main goal of the ongoing project is to understand the processing characteristics and capabilities of RBFs and their visualization aspects. Thanks to our efficient octal tree algorithm, the resulting matrix is a band-diagonal matrix (not simply a sparse one) that reduces computational complexity, allows the application of a simple and direct SLAE solver, and permits the exploration of sufficiently large data sets. C++ language was used to create reusable, extensible, and reliable components, which can be used in later research. The toolkit we have created can be used in conjunction with other algorithms to create animated applications.

Our second goal was very clear: to develop an application for introductory university-level courses for students with no programming and CG experience. Students can download our software binaries (Linux and Windows versions) from [29]. On this page, we have also established an online reconstruction server, where they can input a data file and get a visualization of a VRML object in their browser (Netscape Communicator 4.x is recommended).

6 Acknowledgments

We would like to thank our anonymous reviewers for the comments, which were very helpful and important for our current and further research.

References

[1] V. Shapiro, Real Functions for Representation of Rigid solids, *Computer Aided Geometric Design*, **11**(2): 153-175, 1994.

[2] A. Pasko, V. Adzhiev, A. Sourin and V. Savchenko, Function Representation in Geometric Modeling: Concepts, Implementation and Applications, *The Visual Computer*, **11**(6): 429-446, 1995.

[3] http://www-sop.inria.fr/prisme/fiches/Reconstruction/

[4] B. Geiger, Three-dimensional modeling of human organs and its application to diagnosis and surgical planning, TR2105, INRIA, France, Dec 1993.

[5] V. Savchenko, A. Pasko, T.L. Kunii and A. Savchenko, Feature Based Sculpting of Functionally Defined 3D Geometric Objects, *Proceedings of the Multimedia Modeling Conference*, Singapore, 14-17 Nov., T.T. Chua et al. (eds.), World Scientific Pub., 341-34, 1995.

[6] S. Lee, G. Wolberg, and S.Y. Shin, Scattered Data Interpolation with Multilevel B-Splines, *IEEE Transaction on Visualization and Computer Graphics*, **3**(3), 228-244, 1997.

[7] S. Muraki, Volumetric Shape Description of Range Data Using "Blobby Model", *Computer Graphics (Proceedings of SIGGRAPH)*, **25**(4), 227-235, 1991.

[8] J.H. Ahlberg, E.N. Nilson, J.L. Walsh, *The Theory of Splines and Their Applications*, Academic Press, New York, 1967.

[9] J. Dushon, *Splines Minimizing Rotation Invariants Semi-Norms in Sobolev Spaces*, *Constructive Theory of Functions of Several Variables*, W. Schempp and K. Zeller (eds.), Springer-Verlag, 85-100, 1976.

[10] V.A. Vasilenko, *Spline-functions: Theory, Algorithms, Programs*, Novosibirsk, Nauka Publishers, 1983.

[11] R.M. Bolle and B.C. Vemuri, On Three-Dimensional Surface Reconstruction Methods, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**(1), 1-13, 1991.

[12] G. Greiner, *Surface Construction Based on Variational Principles, Wavelets, Images and Surface Fitting*, P-J. Laurent et al. (eds.), AL Peters Ltd., 277-286, 1994.

[13] V. Savchenko and V. Vishnjakov, The Use of the "Serialization" Approach in the Design of Parallel Programs Exemplified by Problems in Applied Celestial Mechanics, *Performance Evaluation of Parallel Systems, Proceedings PEPS*, University of Warwick, Coventry, UK, 29-30 Nov., 126-133, 1993.

[14] P. Litwinovicz and L. Williams, Animating Images with Drawing, *Computer Graphics (Proceedings of SIGGRAPH)*, 409-412, 1994.

[15] J.C. Carr, W.R. Fright and R.K. Beatson, Surface Interpolation with Radial Basis Functions for Medical Imaging, *IEEE Transaction on Medical Imaging*, **16**(1), 96-107, 1997.

[16] F.L. Bookstein, Principal Warps: Thin Plate Splines and the Decomposition of Deformations, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**(6), 567-585, 1989.

[17] V. Savchenko, A. Pasko, O. Okunev and T.L. Kunii, Function Representation of Solids Reconstructed from Scattered Surface Points and Contours, *Computer Graphics Forum*, **14**(4), 181-188, 1995.

[18] R.K. Beatson and W.A. Light, Fast Evaluation of Radial Basis Functions: Methods for 2-D Polyharmonic Splines, Tech. Rep. 119, Mathematics Department, Univ. of Canterbury, Christchurch, New Zealand, Dec. 1994.

[19] W. Light, *Using Radial Functions on Compact Domains, Wavelets, Images and Surface Fitting*, P. J. Laurent et al. (eds), AL Peters Ltd., 351-370, 1994.

[20] J.C. Carr, T.J. Mitchell, R.K. Beatson, J.B. Cherrie, W.R. Fright, B.C. McCallumm and T.R. Evans, Reconstruction and representation of 3D Objects with Radial Basis Functions, *Computer Graphics, (Proceedings of SIGGRAPH)*, 67-76, 2001.

[21] G. Turk and J.F. O'Brien, Shape Transformation Using Variational Implicit Functions, *Computer Graphics, (Proceedings of SIGGRAPH)*, 335-342, 1999.

[22] V. Savchenko and L. Schmitt, Reconstructing Occlusal Surfaces of Teeth Using a Genetic Algorithm with Simulated Annealing Type Selection, *6th ACM Symposium on Solid Modeling and Applications*, Sheraton Inn, Ann Arbor, Michigan, June 4-8, 2001.

[23] L. Greengard and V. Rokhlin, A Fast Algorithm for Particle Simulation, *J. Comput. Phys.*, **73**, 325-348, 1987.

[24] H. Wendland, Piecewise Polynomial, Positive Defined and Compactly Supported Radial Functions of Minimal Degree, *AICM*, **4**, 389-396, 1995.

[25] B. 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, *Proceedings of the Shape Modeling conference*, Genova, Italy, 89-98, May 2001.

[26] N. Kojekine, V. Savchenko, D. Berzin, and I. Hagiwara, Software Tools for Compactly Supported Radial Basis Functions, *Proceedings of the Fourth IASTED International Conference "Computer Graphics and Imaging"*, Honolulu, Hawaii, USA, 234-239, August 13-16, 2001.

[27] H. Wendland, On the Smoothness of Positive Definite and Radial Functions, *(Preprint submitted to Elseivier Preprint)*, 14 September 1998.

[28] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley Pub Co, 1986.

[29] `http://www.karlson.ru/csrbf/`

[30] A. Jennings, A Compact Storage Scheme for The Solution of Symmetric Linear Simultaneous Equations, *Comput. Journal*, **9**, 281-285, 1966.

[31] G.H. Golub and C.F. Van-Loan, *"Matrix Computations"*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, 1996.

[32] A. George and J.W.H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall: Englewood Cliffs, NJ, 1981.

[33] W.H. Press, S.A. Teukolsky, T. Vetterling and B.P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1997.

[34] W. Schroeder, K. Martin and B. Lorensen, *The Visualization Toolkit. An object-oriented approach to 3D Graphics*, ISBN 0-13-954694-4. Prentice Hall, 1996.

[35] W. Schroeder, K. Martin, L.Avila and C. Law, *The Visualization Toolkit Textbook and open source C++ Library, with Tcl, Python, and Java bindings*, `http://public.kitware.com/VTK/`, published by Kitware 2001.

[36] V. Savchenko and A. Pasko, Reconstruction from Contour Data and Sculpting 3D Objects, *Journal of Computer Aided Surgery*, **1**, 56-57, 1995.

[37] `http://www.research.microsoft.com/~hoppe/`

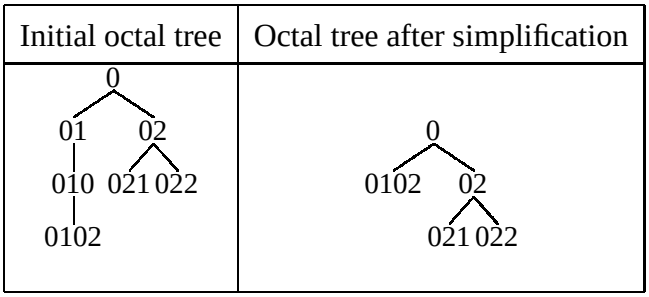


Fig. 1. Octal tree simplification.

	"Head", Fig. 4 (a), number of points $N = 1487$, the selected radius of support $r = 0.2$	"Seashell", Fig. 4 (b), number of points $N = 915$, selected radius of support $r = 0.2$	"Venus", Fig. 4 (c), number of points $N = 6719$, selected radius of sup-port $r = 0.13$
Tree creation time	0.001 sec.	0.001 sec.	0.01 sec.
Sorting time	0.03 sec.	0.03 sec.	0.26 sec.
Matrix calculation time	0.05 sec.	0.02 sec.	0.58 sec.
Memory requirement to store the band diagonal sub-matrix of the matrix A	$1,675,800$ bytes (1 MiB) (if stored traditionally it would be $8,856,576$ bytes (8 MiB))	$669,068$ bytes (0 MiB) (if stored traditionally it would be $3,348,900$ bytes (3 MiB))	$21,171,936$ bytes (20 MiB) (if stored traditionally it would be $180,579,844$ bytes (172 MiB))
Solution time with Cholesky decomposition	0.911 sec.	0.1 sec.	39.01 sec.
Polygonal surface extraction time	0.49 sec.	0.41 sec.	2.73 sec.

Table 1
Processing time. Test configuration: AMD Athlon 1000 Mhz, 128 MB RAM, Microsoft Windows 2000.

```
input_list — unsorted list of points
output_list — sorted list of points
neighbors_ids_list — temporary list of integers

i := 0
while (input_list.length ⩾ 0) do
begin
  // add first element of input_list to output_list
  // remove first element from input_list
  output_list.add(input_list[0]);
  input_list.remove(0);
  while (i < output_list.length) do
  begin
    // find in input_list all neighbors of output_list[i] and
    // put their indices into neighbors_ids_list
    // this can be done with the help of octree
    neighbors_ids_list=
        FindNeighbors(output_list[i],input_list);
    for j := 0 to neighbors_ids_list.length do
    begin
      // add neighbor element of input_list to output_list
      // remove this element from input_list
      output_list.add(input_list[neighbors_list[j]]);
      input_list.remove(neighbors_ids_list[j]);
    end
  end
end
```

Fig. 2. Data sorting to obtain a band-diagonal sub-matrix.

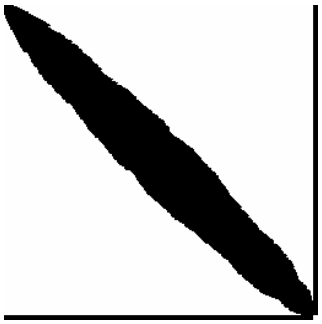
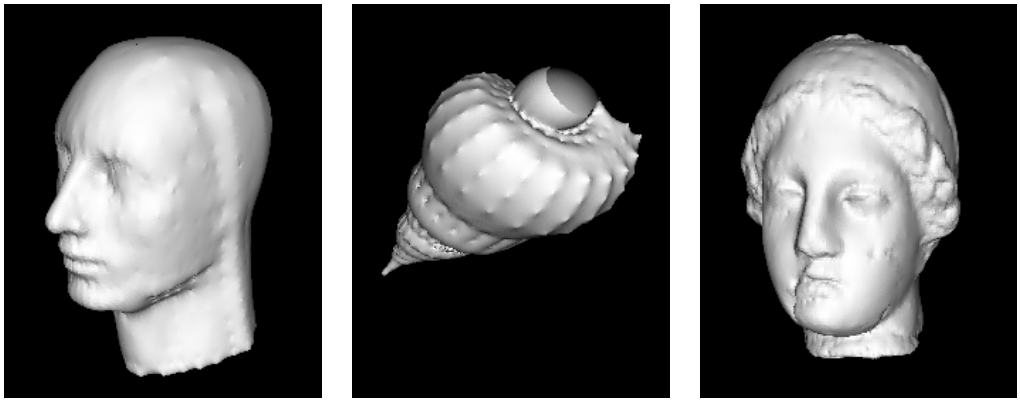


Fig. 3. An example of a typical band-diagonal matrix constructed by the proposed algorithm.



(a) (b) (c)

Fig. 4. (a) The "Head" model reconstruction, (b) The "Seashell" model reconstruction, (c) The "Venus" model reconstruction.

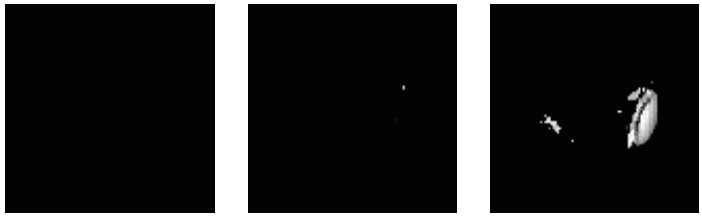


Fig. 5. Visual illustration of the solution as the difference (white area) between RBF and CSRBF solutions with the radii of support 0.4, 0.3, and 0.2, respectively.

	Solution time	Memory required to store matrix A
RBF	20.2 sec.	8 MiB
CSRBF, $r = 0.2$	0.9 sec.	1 MiB
CSRBF, $r = 0.3$	2.2 sec.	2 MiB

Table 2
Processing time and memory requirements for different radii of support. Test configuration: AMD Athlon 1000 Mhz, 128 MB RAM, Microsoft Windows 2000.

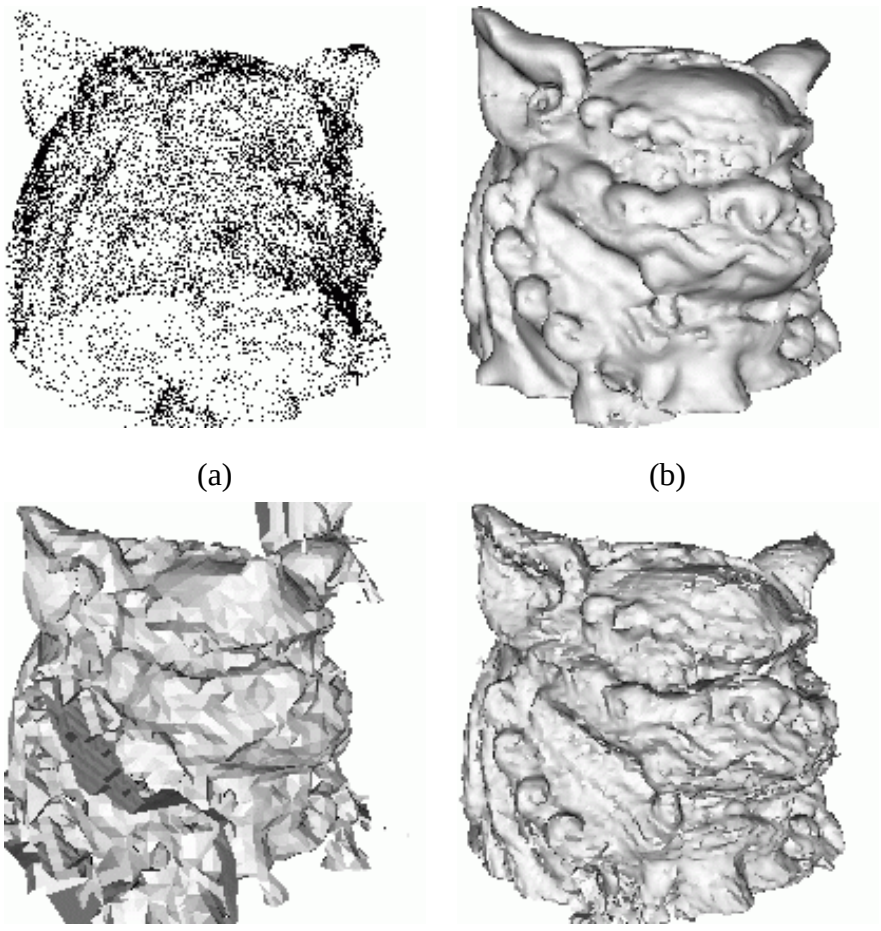


Fig. 6. (a) Source (range data) points of the "Lion-dog" model (courtesy of Dr. A. Belyaev of The University of Aizu), (b) Reconstruction from 19125 points; a sphere is used as a "carrier function" with $r = 0.3$; reconstruction is very slow (about 38 min), due to a large radius of support, but the result looks visually smooth, (c) Reconstruction based on the PhD work of Hugues Hoppe [37], (d) Reconstruction based on the "both-sides" approach; the selected radius of support $r = 0.015$, and 39210 "both sides" constraints are used. The processing time in our test configuration (AMD Athlon 1000 Mhz, 128 MB RAM, Microsoft Windows 2000) is made up as follows: tree build time: 0.04 sec, sorting time: 0.401 sec, matrix constructing time: 1.322 sec, solving time: 12.017 sec. Size of band-diagonal matrix A: 31752656 bytes (30 MiB). Artifacts, which can be observed in the image, were caused by problems of surface extraction.