KURENAI
Kyoto University Research Information Repository

KYOTO UNIVERSITY

| | |
|---|---|
| Title | COMPUTER GRAPHICS APPLICATIONS BASED ON COMPACTLY SUPPORTED RADIAL BASIS FUNCTIONS TOOLKIT (Nonlinear problems in industry, domain decomposition methods and numerical simulations) |
| Author(s) | Kojekine, Nikita; Savchenko, Vladimir; Berzin, Dmitrii; Hagiwara, Ichiro |
| Citation | (2002), 1288: 64-72 |
| Issue Date | 2002-09 |
| URL | http://hdl.handle.net/2433/42493 |
| Right | |
| Type | Departmental Bulletin Paper |
| Textversion | publisher |

Kyoto University

# COMPUTER GRAPHICS APPLICATIONS BASED ON COMPACTLY SUPPORTED RADIAL BASIS FUNCTIONS TOOLKIT

NIKITA KOJEKINE[†], VLADIMIR SAVCHENKO[‡], DMITRII BERZIN[*], ICHIRO HAGIWARA[**].

†, *, ** Faculty of Engineering, Hagiwara's Lab., Tokyo Institute of Technology, 2-12-1, O-okayama, Meguro-ku, Tokyo 152-8552, Japan. E-mail: karlson@stu.mech.titech.ac.jp, dima@mech.titech.ac.jp, hagiwara@mech.titech.ac.jp

‡ Faculty of Computer and Information Sciences, Hosei University, 3-7-2 Kajino-cho Koganei-shi, Tokyo 184-8584, Japan. E-mail: vsavchen@k.hosei.ac.jp

## Abstract

In this paper the use of compactly-supported radial basis functions for surface reconstruction is described. To solve the problem of reconstruction or volume data generation specially designed software is employed. Time performance of the algorithm and numerical error estimation of the reconstruction are also investigated. Thanks to the efficient octree algorithm used in this study, the resulting matrix is a band diagonal matrix that reduces computational cost and permits handling large data sets.

**Keywords**: *surface reconstruction, interpolation, scattered data, volume modeling, computer graphics.*

## 1. Introduction

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 an *implicit model* or 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], and [2]). In [3], an approach to volume modeling is proposed. It combines the voxel representation and function representation. It supposes that the two main representations (voxel/function) are given rather autonomously and a rich set of operations can be used for modeling volumes. Whatever complex operations have been applied to a geometric object, which can be given by a voxel raster, by elevation data, or by a function representation, equivalent volume data, can be generated. Many practical surface reconstruction techniques such as restoration design or reverse engineering tasks based on measured data points require the solution of optimization problems in fitting surface data. We use the term *volume model* to refer to the wide class of surfaces which like other implicit surfaces descriptions can be used for CSG.

A vast volume of literature is devoted to the subject of scattered data reconstruction and interpolation. In most applications, a Delaunay triangulation is used for 3D reconstruction. Unfortunately, this method has some serious drawbacks, even with the elimination of large triangles; the reconstructed shape remains convex-looking, as noted in [4].

Another approach to surface reconstruction is skeletal. An implicit surface generated by point skeletons may be fit to a set of surface points [5], but this method is rather time consuming.

One other approach is to use methods of scattered data interpolation, based on the minimum-energy properties [6], [7], [8]. These methods are widely discussed in mathematical literature (see [9], [10]). The benefits of modeling 3D surfaces with the help of radial basis functions (RBF) have been recognized in [11] for Phobos reconstruction. They were adapted for computer animation [12], [13] and medical applications [14], [15] and were first applied to implicit surfaces by Savchenko et al. [16]. However, the required computational work is proportional to the number of grid nodes and the number of scattered data points. Special methods to reduce the processing time were developed for thin plate splines and discussed in [17], [18].

Actually, the methods exploiting the RBFs can be divided into three groups. 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 [19]). Second group is fast methods for fitting and evaluating RBFs, which allow large data sets to be modeled [20,17]. The third and last group is compactly supported radial based functions (CSRBFs) introduced by Wendland in [21].

In practice, the problem of reconstruction consists of the following steps: sorting the data, constructing the system of linear algebraic equations (SLAE), solving the SLAE, and evaluating the functions. The numerical solution of SLAEs is commonly encountered in various applications. Unfortunately, a "substantial" collection of routines for sparse matrix calculation in [22] could not be found that has forced us to develop our own tool for SLAE solution. 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 contribution, we made an attempt to solve the problem according to the above-mentioned steps. Thus, the main goal of the ongoing project was to develop an effective library of C++ classes that can be successfully applied to computationally intensive problems of surface reconstruction using RBFs splines.

## 2. Method of shape reconstruction with RBF splines

The problem of constructing a smooth reconstructed surface that satisfies certain constraints can be considered as follows. Let $\Omega$ be an n-dimensional domain of an arbitrary shape that contains a set of points $P_i = (x^i_1, x^i_2, \ldots, x^i_n)$: $i = 1,2, \ldots, N$. We assume that the points $P_i$ are distinct and lie on some surface. The goal of the reconstruction is to find a smooth function $f(x_1, x_2, \ldots, x_n)$ that approximately describes the surface. In the three-dimensional case, a volume object is represented by the equation $f(x,y,z) = 0$. The general idea of our algorithm [16] is to introduce a carrier solid with a defining function $f_c$ and to construct a volume spline $U$ interpolating values of the function $f_c$ in the points $P_i$. The algebraic difference between U and $f_c$ describes a reconstructed solid. The algorithm consists of two steps. At 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. The data set $r$ associated with the points $\{r_i = f_c(P_i)$: $i = 1,2, \ldots, N\}$ is then calculated at all given points. In the second step, these values are approximated by a volume spline derived for random or unorganized points. The problem is to construct an interpolation spline function $U(P_i) \in W_2^m(\Omega)$, where $W_2^m$ is the set of all functions whose squares of all derivatives of order $\leq m$ are integrable over $\mathbf{R}^n$, so that $U(P_i) = r_i$, $i = 1,2, \ldots, N$, and has the minimum energy of all functions that interpolate the values $r_i$. This conforms to the following minimum condition (see [8], [10]), which defines operator $T$:

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

where $m$ is a parameter of the variational functional and $\alpha$ is a multi-index. The minimization of the functional results in the nonsingular system of $N + k$ linear algebraic equations for the spline coefficients, where $k = 4$ for the 3D case. After solving for the weight $\lambda_i$ and $v_{0,1,2,3}$ (if a polynomial is required) that satisfy the known constraints $\{r_i = f_c(P_i)$: $i = 1,2, \ldots, N\}$, we can restore the spline-function of the form

$$U(x,y,z) = 1/2 \sum_{i=1}^{N} \lambda_i \phi(|P - P_i|) + p(P), \quad \blacklozenge$$

where $p = v_0 + v_1 x + v_2 y + v_3 z$ is a polynomial of low degree and P is a point with coordinates $(x,y,z)$. The zero set of the function $f(x,y,z) = U(x,y,z) - f_c(x,y,z)$ for $n = 3$ approximates the unknown surface. The isosurface of the 3D object can then be produced. In three dimensions, the thin–plate solution is equivalent to using the radial basis function $\phi(r) = r^3$. Since the function $\phi(r) = r^3$ is not compactly supported, the corresponding SLAE is not sparse or bounded. Storing the lower triangle matrix requires $O(N^2)$ real numbers and the computational complexity is $O(N^3)$. Thus, the amount of computation becomes significant, even for a moderate number of points. Wendland in [21] recently constructed a new class of positive definite and compactly supported radial functions of the form

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

whose the radial of support is equal to 1. For 3D space, we use a $\phi(r) = (1 - r)^2$ interpolated function that supports only $C^0$ continuity. However, other functions that support higher continuity can be used. An investigation [23] of the smoothness of this family of polynomial basis functions shows that each member $\phi(r)$ possesses an even number of continuous derivatives.

## 3. Algorithm for reconstruction

### 3.1 Sorting 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. The structure of octal trees [24] is very similar to that of binary trees, and has been very well studied in the literature (see, for example [25]). 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).

All memory needed to store such a tree is (memory for each node) × (number of nodes) + (memory for storing point) × $N$. In our software implementation, we are employing standard approach for creating the tree from an initial point data set (Algorithm I - "Octal tree creation") with an additional required parametric value $K$, which denotes the maximum number of points in the leaf. Afterwards we can use this tree to search for neighbors of any given point for the given $N$ points. The neighbors are points of a sphere of radius $r$ which origin is located at the given point. We call this sphere an $r$-sphere.

Algorithm II - "Searching using the tree," a "searching" function is used to state for each node in the tree that the node is either:

- (a) entirely inside the given r-sphere,

- (b) entirely outside the given r-sphere,

- (c) partly inside the given r-sphere, partly outside it.

To accelerate the search the tree is simplified after creation. If node $A$ has only one sub-node $B$, we can remove node $A$ and replace it with $B$. For example, consider the following octal tree (numerals represent node numbers):

| Initial octal tree | Octal tree after simplification |
| --- | --- |
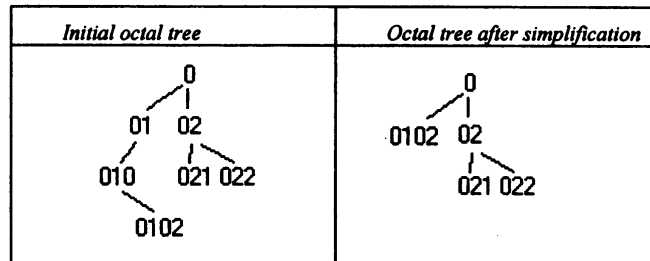


This procedure is needed only if $K$ is small. For instance, for data in the Example 1 (see Table 1. Processing time (in seconds).), if $K$ is set to 1 this procedure removes 113 nodes from the initially created 2427 nodes.

The maximum complexity of the two algorithms that have been described 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 this tree, which is $\log_8 N$. The case in which the depth of the tree would be $N$ is also possible, but it is very improbable (all points would have to belong to the one selected r-sphere). A more detailed account of these algorithms, including a C++ library description can be found in [26]. The procedure of searching for the neighbors of a point in a given $r$-sphere is applied several times in the application. Mainly, it is used for calculating the function ♦ to sum up only the points that are neighbors of the specified point with coordinates $(x,y,z)$. But 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. In our application, to store the band matrix, we use the so-called profile form or a slightly modified version of the Jennings envelope scheme [27]. 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 III - "Sorting data using an octal tree" (Figure 1):

```
input_list          - input list of points
output_list         - output list of points
neighbors_ids_list  - temporary list of integers

i:=0;
while (input_list.lehgth >= 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 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
```

**Figure 1. Pseudo-code of Algorithm III**

As a result of this algorithm a band with maximum size $\alpha_1$ 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 algorithm II). We can reduce our computational outlays by calculating the matrix and the order of the points simultaneously.
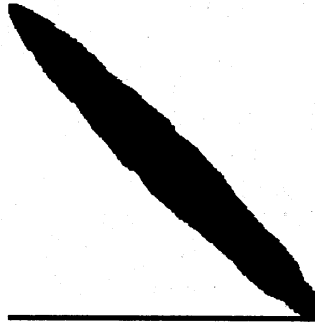


**Figure 2. Typical matrix with band-diagonal part created by algorithm III.**

After sorting we have a banded matrix shown in Figure 2, the matrix has the maximum number of nonzero elements for some point. Naturally, the results of matrix construction depend on the proper selection of the radius of the $r$-sphere. Note that the special order prescribed by sparse matrix to minimize fill-ins is not important. Note also that the half-width for selected $r$ cannot be decreased. Considering the following unlikely event would 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.

| | Example 1. "Head", AMD Athlon 1 Ghz, 128 MB RAM, Windows 98 (*Intel Pentium III 700 MHz, 320 MB RAM, Windows NT*) | Example 2. "Seashell", AMD Athlon 1 Ghz, 128 MB RAM, Windows 98 (*Intel Pentium III 700 MHz 320 MB RAM, Windows NT*) |
|---|---|---|
| Number of points -- $N$ | 1487 (model of the head) | 915 (model of the seashell) |
| Max. number of points in the leaf – $K$ | 10 | 10 |
| Tree creation (inc. file reading time) | 467 nodes at 6 levels. 0.05 (*0.11*) sec. | 291 nodes at 8 levels. 0.05 (*0.07*) sec. |
| Selected radius – $r$ | 0.2 | 0.2 |
| Matrix calculation | *0.17 (0.98)* sec. | 0.1 (*0.2*) sec. |
| Memory requirement to store the band diagonal sub-matrix of the matrix A | 426811 double ≈ 3 MB (if stored traditionally it would be 2211169 double ≈ 16 MB) | 189310 double ≈ 1 MB (if stored traditionally it would be 837225 double ≈ 6 MB) |
| Solution time by Cholesky decomposition | 1.98 (*1.722*) sec. | 0.61 (*0.501*) sec. |
| Root mean square measure (RMS) and maximum absolute error (MAX) | RMS = 1.36945e-008<br><br>MAX = 4.48841e-006 | RMS = 5.8138e-008<br><br>MAX = 3.07849e-005 |
| Result: | Figure 3 (a). | Figure 3 (b). |

**Table 1. Processing time (in seconds).**
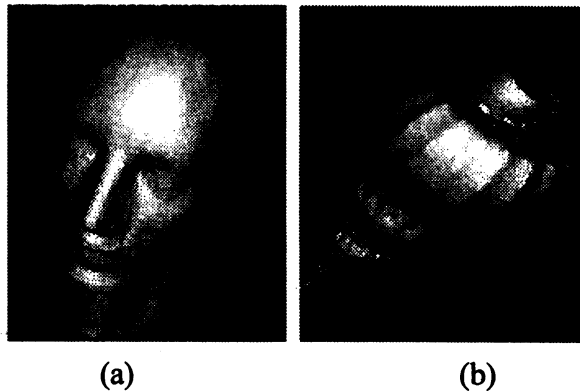


(a)                    (b)

**Figure 3. (a)** "Head" **reconstruction. (b)** "Seashell" **reconstruction**

## 3.2 SLAE solution

Note that solving any sparse system has the goals of saving time and space. The advantage of Gaussian LU [28] decomposition has been well recognized, and many software routines have been developed. For a symmetric and positive definite matrix, a special factorization, called Cholesky decomposition, is about twice as fast compared to alternative methods for solving linear equations. From the discussion in section 3.1 it follows that in the most common case there is a doubly bordered band diagonal system T, which consist of three blocks, square sub-matrices A and D of size $N \times N$ and $k \times k$ respectively, and B, which is not necessarily square and has

the size $N \times k$. A combination of block Gauss solution and Cholesky decomposition was proposed by George and Liu [29] and in our software tools we follow their proposal.

## 3.3 Surface evaluation and extraction

The surface fitted to a set of surface data point forms a volume model of a geometric object. This surface can be visualized directly by using an implicit ray-tracer, and can be voxelized, or polygonized to extract a mesh of polygons. For the visualization of reconstructed volumes an implicit function modeler tool is used (see [30]).
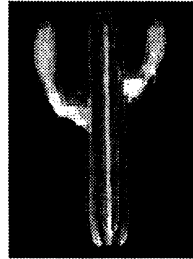


**Figure 4. "Cactus" reconstruction by using the combination of Turk and O'Brien approach and approach presented here.**

Note that the approach taken in this study does not guarantee a restoration of highly topologically complex volume objects (like objects with branching, see Figure 4 for example). For 3D reconstruction using cross-sectional data, in [31] 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 calculates the description of the reconstructed 2D object. For the 3D case such an approach looks exceedingly complicated. Moreover, RBFs demonstrate excessive blending features that lead to undesirable smoothing effects. The approach taken by Turk and O'Brien in [32] of using points specified on both sides of the surface will provide successful restoration of a surface, but it involves drawbacks that leads one to suppose that it would be inefficient for applying RBFs for volume reconstruction. Since it is out of the scope of this paper to discuss all these matters, we would like to mention the following: this approach does not produce CSG-like solids as required in CSG (see [1]); the "both sides" approach has a problem with the surface extraction (a surface extractor can jump outside the band of non-zero points). There is also 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.

## 4. Other applications

The toolkit can be used in conjunction with other algorithms to create animated applications. For example, Figure illustrates the 3D application of combined mappings to produce a visually smooth transformation between the initial "head" shape (genus 0) into the final "torus" shape (genus 1) in the presence of obstacles (not shown). The series of frames shows the process of temporal metamorphosis and spatial transformations defined by the space mapping.
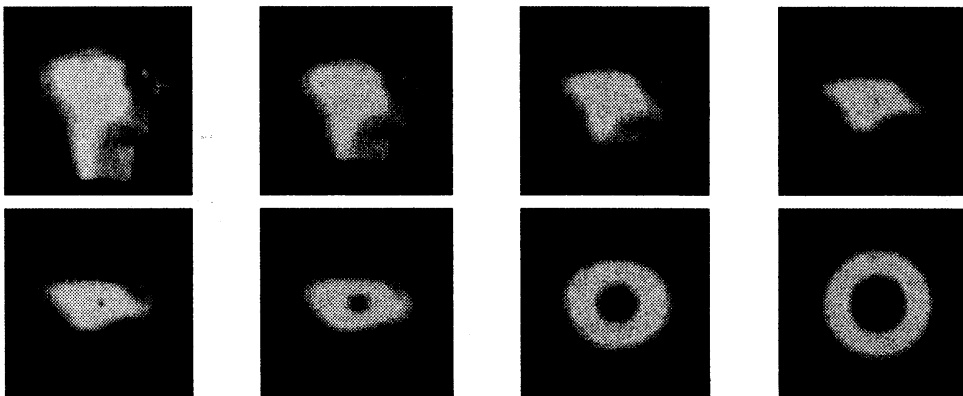


**Figure 5. Metamorphosis with constraints: spatial and temporal transformations.**

It seems also interesting to use the developed CSRBF toolkit with the combination of dynamic programming for the sealing of the holes of the 3D objects. It is a typical task in engineering, occurring for the objects scanned by

laser scanners. For example, object on the Figure 6 was defined by 18355 vertices (3107 borders) and it was processed in only 21,0 seconds by our software.
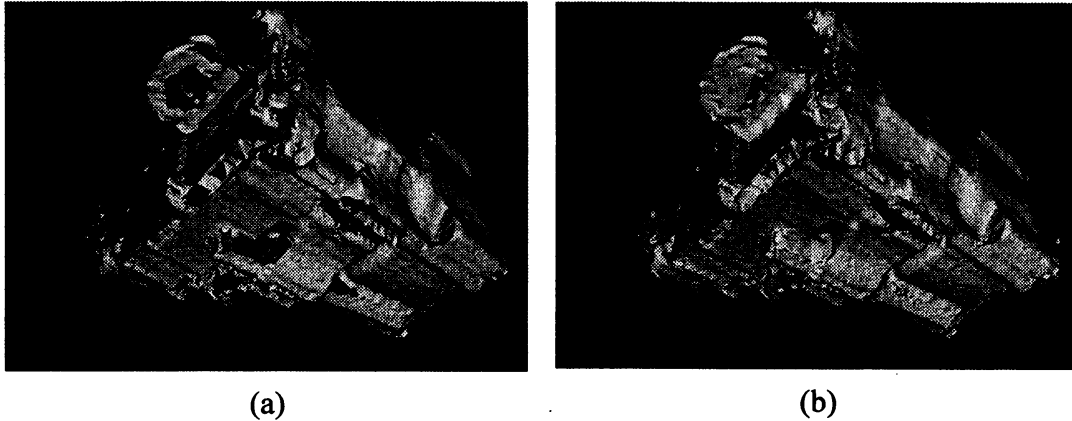


(a)                                   (b)

**Figure 6. Sealing holes in the object: (a) original object, (b) after reconstruction.**

CSRBF functions offer a mechanism to get extrapolated points of a damaged surface, and, actually, exhibit high quality restoration results in surface/image retouching examples. Developed algorithms demonstrate very good results for both black and white images and color images inpainting in small and large areas.

For the example, look at the old photo of president Lincoln on Figure 7. For this example Bertalmio et al. [33] (algorithm based on partial differential equations) reported an inpainting time of approximately 7 min (PII, 300Mhz), while with our software we got inpainting time 0.12 sec on Athlon 1Ghz
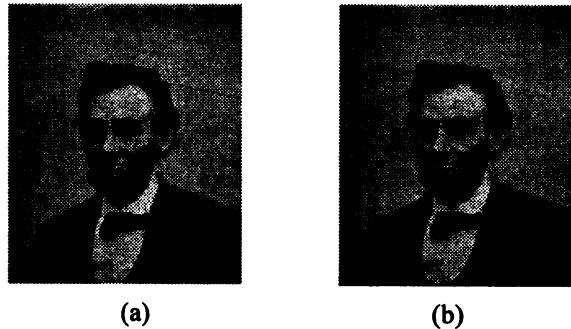


(a)                    (b)

**Figure 7. Inpainting: (a) original photo with defective part, (b) after reconstruction.**

## 5. Results and final remarks

In this contribution, the problem of using RBFs spline reconstruction for volume modeling was thoroughly investigated. The main goal of the ongoing project was to understand the processing characteristics and capabilities of the RBF reconstruction approach and its visualization aspects.

Visual inspection (images in Figure 3 (a), (b)) allows us to judge the interpolation features of the algorithm that has been discussed. Some of the results can be seen in Table 1.

Thanks to the efficient octree algorithm, the resulting matrix is a band diagonal matrix (not a sparse one) that reduces the computational complexity.

An online reconstruction server has been established [26], which makes it possible to get a visualization of a VRML object using a web-browser (Netscape Communicator 4.x is recommended).

Finally, it should be noted that the key concept of "classes" in C++ language might not be ideal for scientific computing. Nevertheless, C++ language was used in this application to create reusable, extensible, and reliable components, which can be used in later 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, V. Savchenko, Function representation in geometric modeling: concepts, implementation and applications, *The Visual Computer*, 11(6), 1995, 429-446

[3] V.V. Savchenko, A.A. Pasko, A.I. Sourin, T.L. Kunii, Volume Modeling: Representations and advanced operations, *Computer Graphics Int. Conf., F. Wolter and N.M. Patrikalakis (eds.)*, Hannover, Germany, IEEE Computer Society, June 22-26 1998, 4-13

[4] S. Djurcilov, A. Pang, Visualizing Sparse Gridded Data Sets, *IEEE Computer Graphics and Applications*, Sept/Oct, 52-57, 2000.

[5] S. Muraki, Volumetric Shape Description of Range Data Using "Blobby Model", *Computer Graphics*, vol.25, no. 4.1991, pp. 227-235 (*Proceedings of SIGGRAPH 91*)

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

[7] 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.

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

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

[10] 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.

[11] V. Savchenko, 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'93*, University of Warwick, Coventry, UK, 29-30 Nov., 126-133, 1993.

[12] P. Litwinovicz, L. Williams, Animating Images with Drawing, Computer Graphics (*Proc. SIGGRAPH'94*), 409-412, 1994.

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

[14] 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.

[15] 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.

[16] V. Savchenko, A. Pasko, O. Okunev and T. Kunii, Function representation of solids reconstructed from scattered surface points and contours, *Computer Graphics Forum*, 14(4), 181-188, 1995.

[17] 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.

[18] 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.

[19] V. Savchenko, 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, (accepted).

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

[21] H. Wendland, Piecewise polynomial, positive defined and compactly supported radial functions of minimal degree, *AICM*, 4, 389-396, 1995.

[22] Intel Math Kernel Library, *Reference Manual*, Copyright 1994-2000, Intel Corporation.

[23] H. Wendland, On the smoothness of positive definite and radial functions, 14 September 1998, *(Preprint submitted to Elseivier Preprint)*.

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

[25] D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley Pub. Co., 1998

[26] http://www.karlson.ru/reconstruction/

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

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

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

[30] *The Visualization Toolkit Textbook and open source C++ Library, with Tcl, Python, and Java bindings*. http://www.kitware.com/vtk.html, published by Kitware 2001.

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

[32] G. Turk and J. F. O'Brien, Shape Transformation Using Variational Implicit Functions, *SIGGRAPH`99*, 335-342, 1999.

[33] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, Image Inpainting, Computer Graphics, SIGGRAPH 2000, July, 417-422, 2000.