Multiscale Implicit Functions for Unified Data Representation

Seongmin Yun and Sanghun Park
Department of Multimedia, Dongguk University
Seoul 100-715 – Republic of Korea
[e-mail: {ysm,mshpark}@dongguk.edu]
*Corresponding author: Sanghun Park

Received July 14, 2011; revised September 29, 2011; revised November 3, 2011; accepted December 7, 2011; published December 31, 2011

Abstract

A variety of reconstruction methods has been developed to convert a set of scattered points generated from real models into explicit forms, such as polygonal meshes, parametric or implicit surfaces. In this paper, we present a method to construct multi-scale implicit surfaces from scattered points using multiscale kernels based on kernel and multi-resolution analysis theories. Our approach differs from other methods in that multi-scale reconstruction can be done without additional manipulation on input data, calculated functions support level of detail representation, and it can be naturally expanded for n-dimensional data. The method also works well with point-sets that are noisy or not uniformly distributed. We show features and performances of the proposed method via experimental results for various data sets.

***Keywords**:* Graphics, multimedia, computer algorithms

This research is supported by Ministry of Culture, Sports and Tourism (MCST) and Korea Creative Content Angency (KOCCA) in the Culture Technology (CT) Research & Development Program 2011, and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2009-0071998).

DOI: 10.3837/tiis.2011.12.007

1. Introduction

The aim of implicit surface reconstruction is to obtain an implicit function from unconstructed scattered data using interpolation or approximation. A surface can then be expressed as the zero level-set of the function. Most software for surface reconstruction is based on radial basis functions (RBFs) or moving least squares (MLSs).

An implicit surface is the zero level-set of a continuous scalar-valued function f that may be an approximation or interpolation of a data set. We need to find a function f that satisfies $f(x_i) = 0, \quad i = 1, \cdots, N$ to reconstruct the implicit surface from a scattered point set $X = \{x_1, \cdots, x_N\}$. In this case, f is usually formed from a combination of basis functions, arranged so that f has positive values inside and negative values outside the object.

Techniques based on RBFs are widely used [1][2][3][4][5][6] due to the accuracy of approximation, flexibility in the choice of basis functions, and convenience in evaluating the approximant. A linear system needs to be solved to find the weights of the RBFs. Carr et al. [1] suggested the so-called fast multipole method that is based on globally supported RBFs but it is hard to implement. Morse et al. [3] used a compactly supported RBF [7] to obtain a sparse interpolation matrix. A MLS approximation is a function that is a solution of a locally weighted least-squares problem [7][8]. This powerful method has been combined with a projection operator to model point-set surfaces [9][10][11]. However, MLS approximation generates a smooth function: to preserve sharp features, Fleishman et al. [12] applied MLS in a piecewise manner, while Lipman et al. [13] used an adaptive spline space in their MLS approximation that incorporates a novel feature detection method for scattered data.

In this paper, we present a data representation technique based on multiscale kernels that have compactly supported, non-symmetric properties [14][15]. Since the kernels support multi-level representation, given n-dimensional data can be converted to implicit functions as unified forms with the level of detail. This approach offers algorithms for various graphics applications, and supports multi-resolution representation that are especially useful for real-time display, adaptive refinement, and progressive transmission. Users can access function values at arbitrary samples from the unified data structures, and get rendered images of the data by traditional graphics techniques. Also, removing insignificant coefficients by thresholding allows to compress the implicit surface. The scheme works with point-sets which are noisy or non-uniformly distributed.

We explain the mathematical background and theories in Section 2. Section 3 describes our reconstruction algorithm, consisting of pre-processing and evaluation stages. Section 4 presents experimental results on volume data as 3D and time-varying volumetric data as 4D. Finally, Section 5 has concluding remarks and outlines future work.

2. Multiscale Kernels

2.1 Mathematical Preliminaries

Kernels are valuable tools for geometric modeling, machine learning and numerical analysis, and their properties have been widely studied in the mathematical literature [7][16]. A kernel is a symmetric function $K : \Omega \times \Omega \to R$, where Ω can be an arbitrary nonempty subset of R^d.

Given a scattered point data set $X=\{x_1,\cdots,x_N\}\subset\Omega$ and a set of values $\{f_1,\cdots,f_N\}$, we want find a continuous function $f^*:R^d\to R$ defined as a linear combination of kernels $f^*=\sum_{i=1}^{N}\alpha_i K(\cdot,x_i)$, where the coefficients α_i are determined by the interpolation condition $f^*(x_j)=f_j,\quad j=1,\cdots,N$. To make this linear system non-singular, the kernel K has to be positive definite so that for every finite set $\{x_1,\cdots,x_n\}$ of pairwise distinct points in Ω, the matrix $(K(x_i,x_j))_{1\le i,j\le n}$ is positive definite. In general, radial kernels, termed RBFs, are widely used, because they are stable under translation and rotation of the point-set.

In this paper, we use the multiscale kernel (MSK) proposed by Opfer [17]. Let the function $\varphi:R^d\to R$ be bounded, compactly supported, and refinable, so that there is a sequence $\{h_k\}_{k\in Z^d}$ of real numbers for which $\varphi=\sum_{k\in Z^d}h_k\varphi(2\cdot-k)$. Let $\ell\in Z$ be a fixed integer and $\lambda:=\{\lambda_j\}_{j=\ell}^{\infty}$ be a sequence of weights with $\lambda_j>0$ and $\sum_{j=\ell}^{\infty}\lambda_j<\infty$. Then the function $\Phi_\ell(x,y):=\sum_{j=\ell}^{\infty}\lambda_j(\sum_{k\in Z^d}\varphi(2^j x-k)\varphi(2^j y-k))$ is a multiscale kernel and it is positive definite. Some restrictions are required to use this kernel in practice. Consider a finite multiscale kernel Φ_ℓ^u, such that

$$\Phi_\ell^u(x,y):=\sum_{j=\ell}^{u}\lambda_j(\sum_{k\in Z^d}\varphi(2^j x-k)\varphi(2^j y-k)), \tag{1}$$

where the refinable function φ is chosen, so that the mask $\{h_k\}$ is finite (e.g. the tensor product of univariate B-splines). Then Φ_ℓ^u is compactly supported, and for $X=\{x_1,\cdots,x_N\}\subset\Omega$ with

$$\min_{i\ne j}\left\|x_i-x_j\right\|_2>r2^{-u+1}, \tag{2}$$

the matrix $(\Phi_\ell^u(x_i,x_j))_{1\le i,j\le N}$ is positive definite, where $supp(\varphi)\subset\{x\in R^d:\left\|x-c\right\|_2<r\}$, and r is the radius of support of function φ.

The advantage of a multiscale kernel lies in its structure. Since φ is refinable, the interpolant f^*, defined as a linear combination of multiscale kernels, possesses a wavelet-like multiresolution decomposition that permits fast evaluation:

$$f^*=\sum_{i=1}^{N}\alpha_i\Phi_\ell^u(x_i,\cdot)=\sum_{j=\ell}^{u}(\sum_{k\in Z^d}c_k^j\varphi(2^j\cdot-k))=:\sum_{j=\ell}^{u}s_j, \tag{3}$$

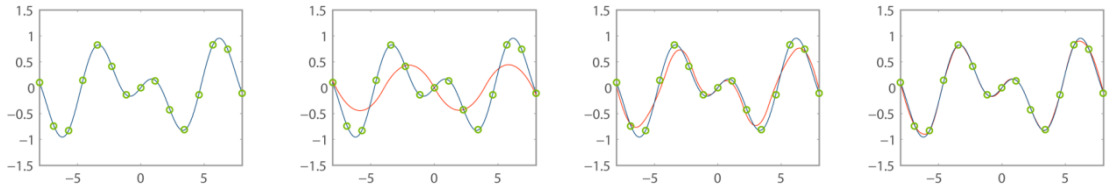
where the coefficients c_k^j are given by

$$c_k^j:=\lambda_j\sum_{i=1}^{N}\alpha_i\varphi(2^j x_i-k),\quad \ell\le j\le u,\quad k\in Z^d. \tag{4}$$

The roughest approximation to f^* is s_ℓ, while the terms s_j with higher values of the index j provide successively more detailed representations of f^*.

For example, consider a function $f(x)=\cos(x)\sin(x)$ and 15 sampled points (see **Fig. 1-(a)**). We use a quadratic B-spline φ with the weights $\lambda_j=1/4^j$, $j=0,\cdots,2$ in Equation (1). Then we have a MSK interpolant $f^*=\sum_{i=1}^{15}\alpha_i\Phi_0^2(x_i,\cdot)=\sum_{j=0}^{2}s_j$ (**Fig. 1-(d)**). Moreover, s_0 is the

roughest approximant to the test function, and $s_0 + s_1$ is a finer approximant than s_0.



(a) $f(x) = \cos(x)\sin(x)$ **(b)** s_0 **(c)** $s_0 + s_1$ **(d)** $s_0 + s_1 + s_2$

Fig. 1. Test interpolant evaluated in 1D.

Since $\Phi_\ell^u(x,\cdot)$ is compactly supported, $(\Phi_\ell^u(x_i, x_j))_{1\le i,j\le N}$ is a sparse matrix and so the coefficients α_i can be calculated using an appropriate numerical method. Moreover, for each $k \in \{k \in Z^d : c_k^u \neq 0\}$, there exists a unique $x_i \in X$ such that $\varphi(2^u x_i - k) \neq 0$. Using a *k*-dimensional tree search, we can calculate this x_i in $O(N \log N)$ operations, and so c_k^u is easily obtained. For $\ell \le j \le u-1$, c_k^j is given by the same equations

$$c_k^j := \frac{\lambda_j}{\lambda_{j+1}} \sum_{\mu \in Z^d} h_{\mu-2k} c_\mu^{j+1} \tag{5}$$

that underlie the fast wavelet transform algorithm. After an initialization process that requires $O(N \log N)$ operations, the function f^* of Equation (3) can be evaluated in $O(1)$ operations. Since the magnitude of λ_j is less at higher levels, contribution of λ_j is low. When our method is used, the choice of appropriate λ_j is critical for multiresolution representation. In addition, there is a limitation in that reconstruction results for each level are not continuous, because multiresolution representation depends on integer j.

3. Reconstruction Algorithms

3.1 Overview

Our scheme consists of two stages: pre-processing and evaluation. The goal of the pre-processing stage is to determine the coefficients α_i and c_j^k of a MSK interpolant from the given n-dimensional data set. The entire input domain is subdivided into small subdomains hierarchically by the kd-tree structure to do this. Then, we construct and solve linear systems based on multiscale kernel for each subdomain. According to properties of multiscale kernels, appropriate data structures and numerical techniques should be implemented to search nonzero elements from the sparse matrix and to solve the linear system effectively since the linear systems compose sparse matrices. Once coefficients of the implicit functions for the highest detail level are found, coefficients for the other levels can be simply calculated by Equation (5). Well-designed data structures are needed to store and extract these coefficients, since they are sparse in integer grids.

Once the multiscale implicit functions for each subdomain are constructed, function values at arbitrary position in overall domain can be evaluated by partition of unity method that calculates a reconstructed function value by weighted sum of local functions for neighbor

subdomains. Users can control the level of detail by choosing an appropriate level for the coefficients used.

Fig. 2 shows the pseudocode to represent the overall processing. In general, the pre-processing stage requires an appreciable amount of computation (line 2∶9); but an evaluation of the function value at an arbitrary n-dimensional point can be done much faster than can the pre-processing steps (line 10).

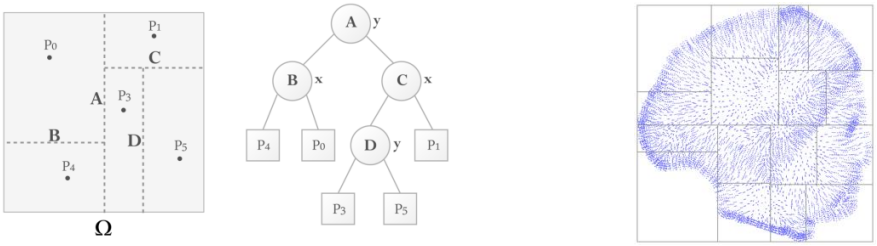
```
1:   procedure MLTSCL_KRNL_SRFC({(x_i, f_i)})
2:     m ← Decompose the domain Ω
3:     for k ← 1, m do
4:       Build up the linear system for f_k (Equation (3))
5:       Solve the linear system (Equation (3))
6:       Calculate the coefficients (Equation (4))
7:       F ← F + f_k
9:     end for
10:    Ray-trace or polygonise F with LOD
11:  end procedure
```

Fig. 2. Pseudo code for multiscale surface reconstruction.

3.2 Calculating Implicit Functions

We partition the given data into smaller subsets by using partition of unity (POU) to deal with large input data on computers with limited memory space. In the POU method, the global domain Ω is divided into small subdomains Ω_i, such that $\Omega \subset \cup_{i=1}^{m} \Omega_i$, with slight overlaps among the subdomains. It is necessary to overlap them to blend the local reconstructed functions [18]. We can then associate a POU with these subdomains, in the form of a family of non-negative, continuous weight functions w_i, such that $supp(w_i) \subset \Omega_i$ and $\sum_{i=1}^{m} w_i(x) = 1$ for all $x \in \Omega$. For each Ω_i, we construct a local approximation f_i, and the global approximant on Ω is formed by weighting these local approximants as follows:

$$F(x) = \sum_{i=1}^{m} w_i(x) f_i(x). \tag{6}$$



(a) kd-tree based hierarchical space partition

(b) a cutting plane after domain decomposition of igea 3D point data

Fig. 3. Hierarchical domain decomposition in the processing.

The weight function w_i can be found by normalization of smooth function W_i, as follows:

$$w_i(x) = \frac{W_i(x)}{\sum_j W_j(x)} \tag{7}$$

such as W_i is continuous at the boundary of subdomain Ω_i.

The kd-tree is used to decompose the domain, because it is applicable to n-dimensional data (**Fig. 3-(a)**). We make recursive subdivision of the kd-tree stop when each subdomain includes less than a certain number of points. For example, igea data with 24,804 points was partitioned into 32 subdomains when we set each subdomain to have at most 1,000 points (**Fig. 3-(b)**). In fact, each domain has about 775 points and the standard deviation is approximately 10.

After partitioning the domain Ω into subdomain Ω_i, a linear system $A\alpha = Y$ is built up for α by Equation (3) where Ω_i has n points:

$$\begin{pmatrix} \Phi_l^u(x_1,x_1) & \cdots & \Phi_l^u(x_1,x_n) \\ \vdots & \vdots & \vdots \\ \Phi_l^u(x_n,x_1) & \cdots & \Phi_l^u(x_n,x_n) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}. \tag{8}$$

Matrix A is sparse symmetric matrix, since the multiscale kernel is a compactly supported and symmetric function. The only elements satisfying $\Phi_l^u(x_i,x_j) \neq 0$ should be calculated to minimize the cost of composing matrix A. We need to find x_j included in the multiscale kernel support $2^{-r} \times 0.5\, r$ where r is the radius of support of function φ. The kd-tree and PARDISO [19], which was developed to find a solution of sparse linear system, and based on LU decomposition, were used for efficient searching and solving the linear system, respectively. We extended PARDISO to an enhanced parallel solver on shared memory multiprocessors.

After finding solutions of the linear systems, we can calculate the coefficients $\{c_k^j : k \in Z^d, l \le j \le u-1\}$ defined in Equation (5) from higher level $u-1$ to lowest l. In each level j, the number of coefficients satisfying $c_k^j \neq 0$ is Cn where $C := \max_{x \in R^d} \{k \in Z^d : \varphi(x-k) \neq 0\}$, and n is the number of data points used in reconstruction. The coefficients in the highest level u are generated by Equation (4) for k, such as $c_k^u \neq 0$:

$$c_k^u = \lambda_u \sum_{i=1}^{n} \varphi(2^u x_i - k).$$

Since $supp(\varphi) \subset \{x \in R^d : \|x-c\|_2 < r\}$ and $(2^u r) < 0.5 \min_{i \neq j} \|x_i - x_j\|$, $x_i \in X$ is unique such as $\varphi(2^u x_i - k) \neq 0$. The other coefficients c_k^j of lower levels j where $l \le j \le u-1$, can be generated recursively by Equation (5).

These coefficients are sparse data defined on integer grids, and are needed to evaluate function values at arbitrary positions. We used hash tables as a data structure to extract the coefficients randomly. In this table, the key is index $k \in Z^d$ of integer grids in d-dimension, and return value is a nonzero coefficient c_j^k.

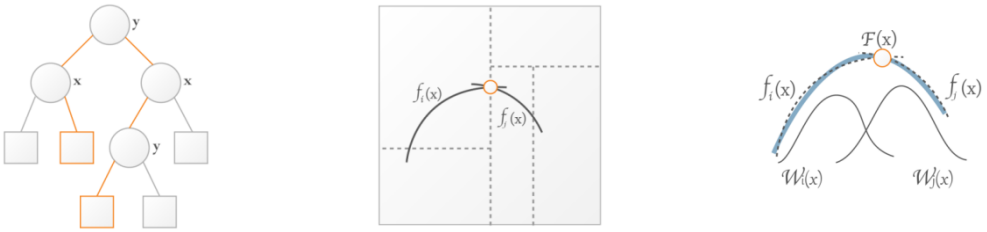
3.3 Evaluating Function Values

We find all subdomains $\{\Omega_i\}$ including x by searching boundary volume hierarchy to evaluate a function value at an arbitrary position x (**Fig. 4-(a)**). Then, the function values

$f_i(x)$ can be calculated from each local implicit function defined in $\{\Omega_i\}$ (**Fig. 4-(b)**). The final value is the weighted sum of local function values: $F(x) = \sum_i w_i(x) f_i(x)$ (**Fig. 4-(c)**). We use decay functions in the partition of unity proposed by Tobor et al [18] as weight functions.

These evaluation steps are done only for the indices k, such that $\varphi(2^j x - k) \neq 0$, to enhance run-time processing speed. The indices k satisfying the above condition were used for the hash tables keys. In addition, the function $f = \sum_{j=l}^{u} s_j$ is a complete interpolant. We can control LOD by modifying the upper bound, as follows:

$$\tilde{f} = \sum_{j=l}^{\bar{u}} s_j, \quad l \leq \bar{u} < u.$$



(a) domain search from the hierarchical structure **(b)** calculating local reconstructed function values **(c)** weighted summation of function values

Fig. 4. Evaluation step for finding a function value.

Fig. 5 shows the results of a simple test visualization in which a multi-resolution representation of digital elevation model (DEM) data with 256×256 resolution was produced by an MSK interpolation, based on the tensor product of quadratic B-splines with $\lambda_j = 2^{(2-2\times1.8)\times j}$, $l = -3, u = 0$. Multiresolution reconstruction was done with four levels, and according to Euclidean distance between camera and evaluation point, the degree of detail can be selected automatically. Surface normals used in rendering were calculated from the gradient of the implicit functions. This program ran in real-time.

4. Experimental Results

4.1 Unorganized Point-sets

In this section, we explain how to create implicit surfaces from scattered data sets, such as igea (8,268 points), bunny (35,947 points), and squirrel (9,995 points). If only given points are used for surface reconstruction, the linear system in Equation (8) has just a trivial solution. To avoid this problem, the points, which have an appropriate offset to normal direction from original points and assigned function values -1 and 1, are used as additional input data. If the original number of points is n, then the number of processed points is $3n$. We subdivided 3D space to make the leaf node in the kd-tree to have less than 1,000 points, each corresponding domain was overlapped with adjacent domains at about 16% ratios. A local implicit function is blended with neighbor functions in C^2 continuity by partition of unity. **Table 1** shows the parameters used in the experiments. Function φ is the tensor product of cubic B-splines. The parameter λ_j should be determined for reconstructed functions to support level of detail representation experimentally. The data dependent parameters, u and l, should be carefully

chosen to make the linear system a positive definite and sparse matrix.

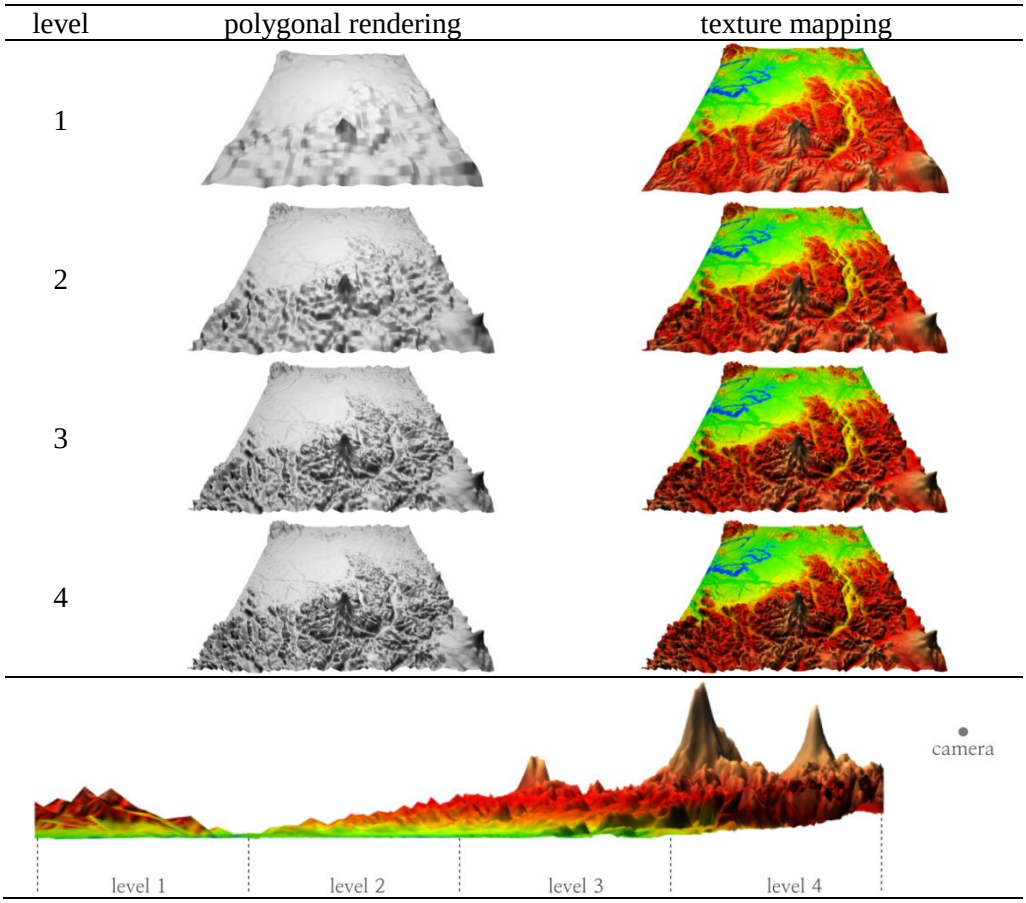


Fig. 5. Multiscale reconstruction of 256×256 DEM data.

Table 1. Parameters used for test 3D scattered point-sets reconstruction experiments and pre-processing times for order of B-spline in seconds.

	pre-processing parameter			order of B-spline		
	l	u	$\lambda_j = 2^{(3-2\times s)\times j}$	1	2	3
igea	4	13	$s = 2.6$	12	28	60
squirrel	4	13	$s = 1.6$	7	22	61

The choice of u of Equation (1) is a critical issue in the application of MSKs. If we use too small u, then the condition expressed by Equation (2) is not satisfied and the interpolation matrix is singular. However, if u is too large, then there is inadequate support for MSK and the interpolant will be of low quality. We introduce a scaled multiscale kernel to overcome this problem, as follows:

$$\Phi_{\ell}^{u}(x/s, y/s) = \sum_{j=l}^{u} \lambda_j \Big(\sum_{k\in Z^d} \varphi(2^j x/s - k)\varphi(2^j y/s - k) \Big).$$

Scale parameters s are determined to guarantee the nonsingularity of the interpolation matrix, as follows:

$$s = 2^{u-1} \min_{i \neq j} \left\| x_i - x_j \right\|_2 / r, \tag{9}$$

where $supp(\varphi) \subset \{x \in R^d : \left\| x - c \right\|_2 < r\}$.

After a sampled volume data with uniform grid structure was created by evaluating function values $f(p)$ for all voxel points $p = (x, y, z)$, an isosurfacing algorithm, such as Marching Cube, was used to compute an isosurface from the volume. **Fig. 6** shows images of surfaces reconstructed from the igea and bunny data sets at different levels of detail (*level* = 2, 3, 4, and 6). Our method differs from the other techniques, such as [4][18], in that any manipulation operations, such as selection, and deletion, are not necessary on input data to create multi-level implicit functions. However, few artifacts appear on rendered surfaces of the lowest level at the boundary of subdomains. The higher the level, the more similar are the rendered images, because the weight λ_j approaches zero as the level increases. Thus, an algorithm that automatically finds optimal λ_j should be developed.

In general, the pre-processing time is proportional to the number of points and the order of the B-spline (see **Table 1**). Although we implemented the pre-processing algorithm without sophisticated optimization techniques such as exploiting GPU or parallelization, it takes about 1 minute (igea, squirrel) or less than 6 minutes (bunny) for cubic B-spline.

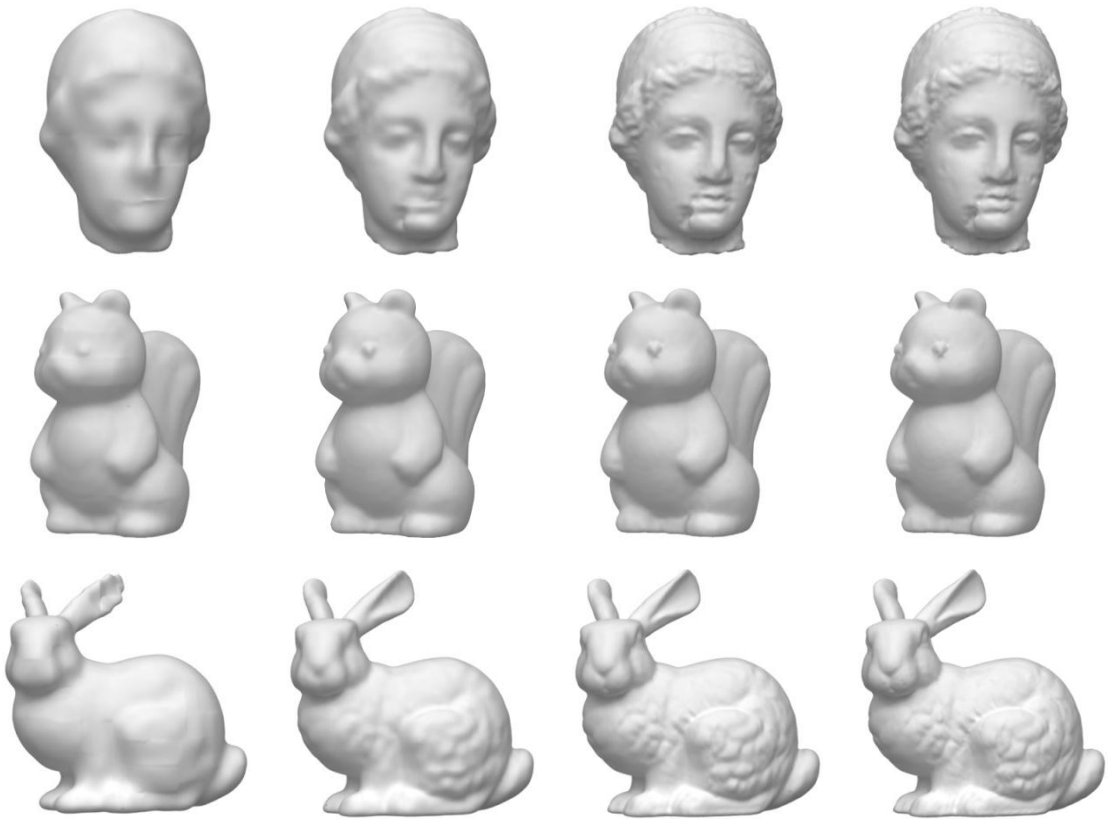
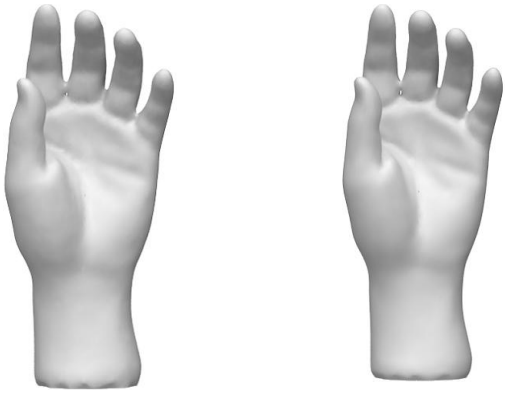


Fig. 6. Surface reconstruction from scattered 3D point data at four different levels of detail (level = 2, 3, 4, and 6).

Table 2 shows time performances for evaluation at sample points in appropriate resolution grids to generate a volume to be isosurfaced. The experimental environment is a general purpose Windows desktop with Intel Core i7 2.80 GHz, and 4.0 GB memory.

Table 2. Evaluation times from implicit function in seconds (Intel Core i7 processor, 2.8 GHz: main memory, 4 GB).

	level of detail					
	1	2	3	4	5	6
igea ($126\times128\times91$)	2	4	5	7	8	10
squirrel ($85\times128\times117$)	2	3	5	6	7	9
bunny ($128\times126\times102$)	4	6	8	10	12	15



(a) MSK **(b)** FastRBF

Fig. 7. Comparison of rendered images of surfaces generated by MSK and FastRBF.

We analyzed our method's timing performance compared to the FastRBF toolbox that implemented RBF by the fast multipole method [1]. Two timing performances were considered on hand data with 15,000 scattered points: (1) reconstruction of implicit functions, and (2) evaluation of function values. All points in the given data were used in surface reconstruction without the center reduction technique for surface reconstruction by FastRBF. The multiscale kernel Φ_4^{12} created by the tensor product of quadratic B-spline is used for our method. The pre-processing times of two methods are 23.8 (MSK) and 11.5 (FastRBF) seconds, so FastRBF is twice as fast as MSK. We harnessed Intel Threading Building Blocks [20] for multi-core programming to enhance CPU processing speeds (it takes 49.4 seconds to pre-process the same data without multi-core programming). In addition, if the state-of-the-art techniques such as sparse linear solvers [21], real-time hashing [22], and parallel k-d trees [23] implemented on high-performance GPUs are employed in our pre-processing stage, the timing performances would be remarkably improved.

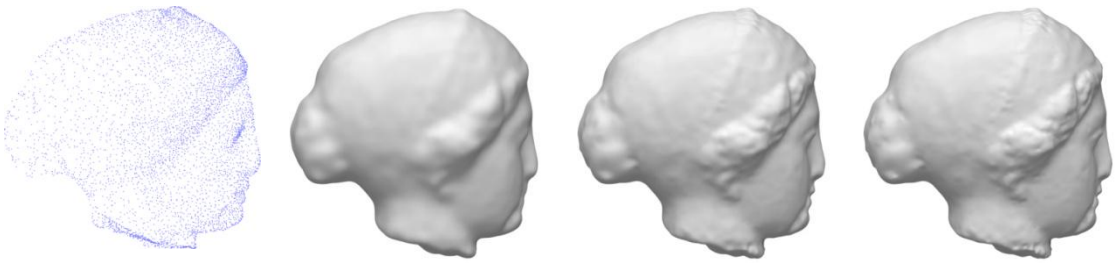
Conversely, the evaluation times of function values in our method are better than for FastRBF (see **Table 3**). Theoretically, both methods can be evaluated in $O(1)$ operations after a preliminary computation requiring $O(N \log N)$ [1][17]. **Fig. 7** shows the rendered images of surfaces evaluated by the methods. It is difficult to notice the differences between them. The maximum reconstruction errors $max\|y_i - f(x_i)\|$ between original input and evaluation points are 4.87×10^{-4} (MSK) and 6.31×10^{-4} (FastRBF).

MSK interpolation performs poorly on a non-uniform data set if its radius of support is chosen

globally. A non-uniform data set can be handled by MSKs with a different radius in each cell of a partition: the scale parameter s can then be determined from Equation (9) using points x_i from each cell. **Fig. 8-(a)** shows the modified non-uniform data set of igea. The sample data set was created by removing randomly selected points, a linear function proportional to the positive direction of an axis was used to control the number of points to be eliminated. Many small partitions tend to concentrate in the regions where the density of points is relatively higher. Conversely, the parts where point distribution is sparser are divided by some bigger partitions. Appropriate kernels with different radii of support for each partition can be dynamically selected. **Fig. 8-(b)** to **Fig. 8-(d)** show surfaces reconstructed in different levels. Even though the given data set is non-uniform, the rendered images are sufficiently good.

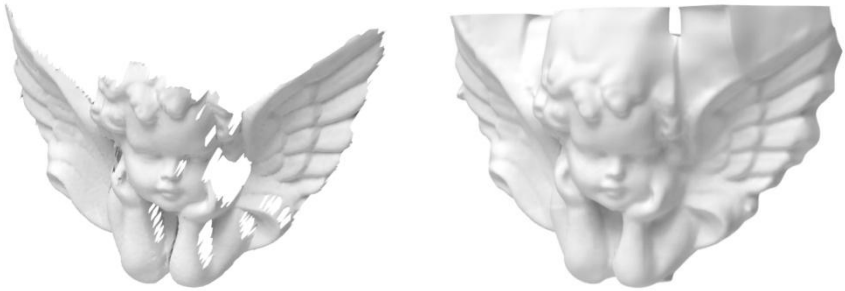
Table 3. Comparison of evaluation times between MSK (level 8) and FastRBF in seconds on hand scattered point data (S: $29\times64\times26$, M: $58\times128\times51$, L: $115\times256\times101$).

	resolution of samples		
	S	M	L
FastRBF	2.2	14.9	115.4
MSK	0.3	2.0	14.2



(a) non-uniformly distributed input points **(b)** level 2 **(c)** level 3 **(d)** level 6

Fig. 8. Multiscale surface reconstruction from non-uniform scattered points.



(a) incomplete mesh with holes (24,566 vertices) **(b)** hole-filling result by our method

Fig. 9. Reconstruction of polygonal mesh with partially missing parts.

Our method can be applied to reconstruct an incomplete polygonal mesh. **Fig. 9-(a)** and **9-(b)** show an input test mesh of 24,566 vertices with partially missing polygons, and the resultant images after applying our surface reconstruction method, respectively. In this experiment, the parameters are as follows: $\lambda_j = 2^{(2-2\times1.6)\times j}, l = 3, u = 16$, and φ is a tensor product of cubic B-spline. Larger supports of reconstruction functions can interpolate wide

areas and finally the holes are filled, since the lower limit l is set a relatively small value. However, that causes the linear system to be dense, as well as increasing computational costs.

We added noise to the bunny data set with perturbation intentionally. **Fig. 10** shows the test data and reconstructed surfaces. The rendered surface is clearly affected by noise, but the silhouette, features are described adequately, and the reconstructed surfaces exhibit the characteristics of noisy data. The de-noising effect in reconstructed surfaces at level 4 (**Fig. 10-(b)**) is better than that in surfaces at level 6 (**Fig. 10-(c)**). An appropriate medium-level reconstruction of implicit surfaces created from noisy data set can be much more useful, especially if the model appears in a small portion of the entire displayed image.

(a) test data with noise

(b) lower level showing de-noising effect on (level 4)

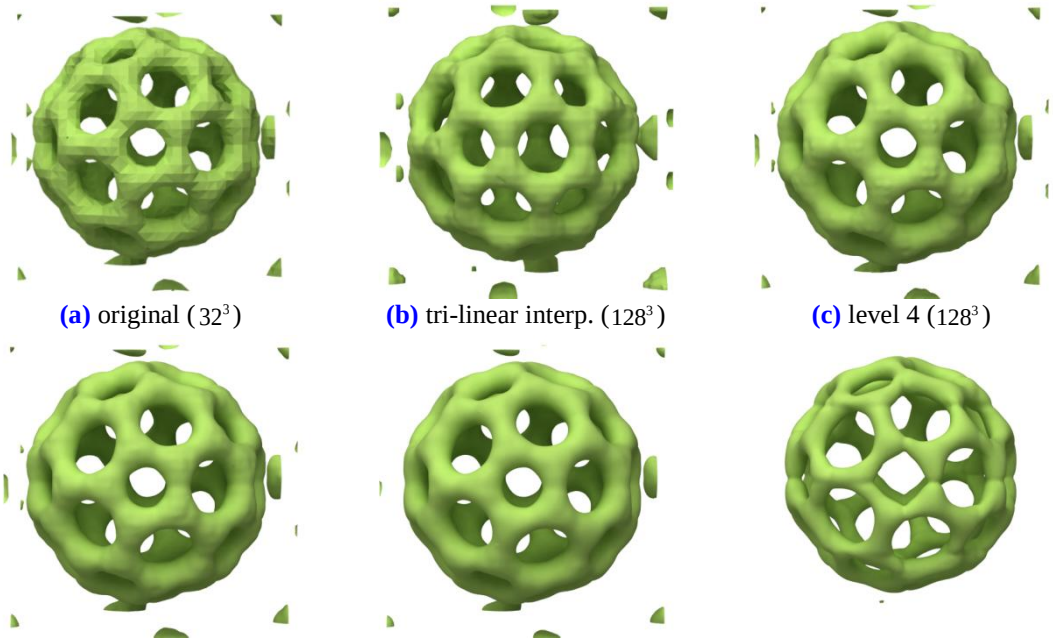
(c) higher level with noticeable noise (level 6)

Fig. 10. Surface reconstruction from scattered points with noise.

4.2 3D Volumetric Data Sets

Volume visualization extracts and renders meaningful information from volumetric data consisting of function values f_v at voxel grids (i, j, k). In general, the volume rendering process requires access of function values $f(p)$ at arbitrary sample points p; they can be calculated by various interpolation techniques. In this section, we demonstrate that our method can efficiently find interpolated function values at any points, and much better rendered images can be generated. We rendered buckyball volume data defined on a 32^3 uniform voxel grid using Marching Cube algorithm. **Fig. 11-(a)** and **(b)** show isosurfaces for an isovalue of 0.5, and the vertices of the surfaces were generated from trilinear interpolation on 32^3 and 128^3 grids respectively. We applied our surface reconstruction method to the volume data with voxel coordinates (i, j, k) and function values f_v. Multiscale kernel Φ_{-1}^2, which consists of cubic B-spline tensor product, was used for this experiment. After finding function values from reconstructed implicit functions at 128^3 sample points on uniform grids, we can render isosurfaces from the new volume (See **Fig. 11-(c)-(f)**). We can find from the images that interpolation based on multiscale kernel is much better than commonly used trilinear methods, and the rendering results show little aliasing. The difference in quality is due to the trilinear interpolation considering only eight voxel values surrounding a sample point, while in contrast, multiscale based implicit functions are created by all voxel values included in the kernel support used. Furthermore, the level of detail representation can be useful in various traditional volume rendering techniques, such as ray-casting and splatting. In addition, if our method is applied to 3D volume data defined on a non-uniform irregular grid, which requires an additional process or complex handling, the user can effectively visualize the data by randomly accessing the function values at any point. The rendered images in **Fig. 12** were created for engine volume data of $64\times64\times32$ resolution under similar conditions used to visualize buckyball in **Fig. 11**. The pre-processing times for buckyball and engine volume data are 41 and 179 seconds respectively, **Table 4** evaluates performance for the access function

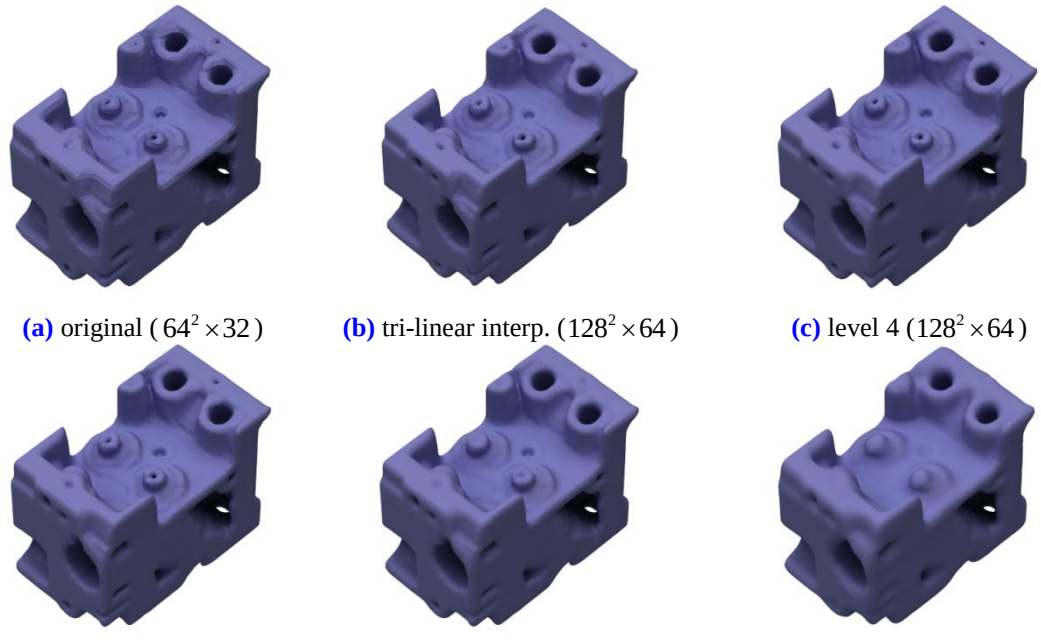
values.



(a) original (32^3) **(b)** tri-linear interp. (128^3) **(c)** level 4 (128^3)

(d) level 3 (128^3) **(e)** level 2 (128^3) **(f)** level 1 (128^3)

Fig. 11. Multiscale reconstruction of a volume data buckyball (32^3).



(a) original ($64^2 \times 32$) **(b)** tri-linear interp. ($128^2 \times 64$) **(c)** level 4 ($128^2 \times 64$)

(d) level 3 ($128^2 \times 64$) **(e)** level 2 ($128^2 \times 64$) **(f)** level 1 ($128^2 \times 64$)

Fig. 12. Multiscale reconstruction of a volume data engine ($64^2 \times 32$).

Table 4. Evaluation times in seconds and maximum errors as a measure of interpolation quality from our multiscale function of the test volume data.

		level			
		1	2	3	4
Evaluation times (in seconds)	buckyball (128^3)	5.5	12.1	18.1	22.7
	engine ($128^2 \times 32$)	2.6	5.4	8.3	12.4
Maximum errors ($max\|y_i - f(x_i)\|$)	buckyball (32^3)	5.7×10^{-1}	2.9×10^{-1}	8.2×10^{-2}	5.6×10^{-6}
	engine ($64^2 \times 32$)	2.7×10^{-3}	1.6×10^{-3}	6.7×10^{-4}	2.3×10^{-8}

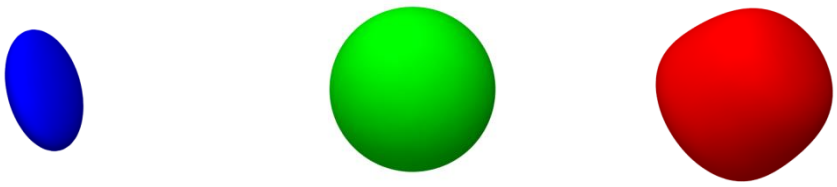
4.3 Time-varying Data Sets

It is natural that our technique can be applied to surface representation of n-dimensional data, since multiscale kernels support n-dimensional data interpolation and approximation. In this section, we show that our method can be useful for 4D data, such as time-varying 3D volumetric data set.

We constructed three volume data sets, which have voxels with function values $f(x,y,z)$ from quadratic functions in **Table 5** at uniform grid points (x,y,z) of 64^3 where $-1.2 \le x,y,z \le 1.2$, for this test. We added additional timing parameter t to the three volumes, and assigned timestep values 0.0, 1.0, 2.0 to t respectively, to make the data time-varying. **Fig. 13** shows isosurfacing results with $f(x,y,z,t)=0$ at the three timesteps. After creating 4D implicit functions from time-varying volume using multiscale kernels Φ^2_{-1} from tensor product of linear B-spline, a function value $f(x,y,z,t)$ at an arbitrary point (x,y,z) and time t can be accessed in the same manner. We can make a morphing animation by evaluating the function values at fixed sample points for $t = t + \Delta t$ $(0.0 \le t \le 2.0)$. **Fig. 14** shows that the object can be interpolated smoothly from 4D function $f(x,y,z,t)=0$ in timing axis t. In this case, the colors of the object were interpolated according to t. It took about 400 seconds to pre-process the three volumes (3×64^3), **Table 6** shows the timing performance to evaluate function values at 64^3 sample points.

Table 5. Functions for test time-varying volume data.

time t	function $f(x,y,z)$
0.0	$x^2/1.5^2 + y^2/1.0^2 + z^2/0.5^2 - 0.6^2$
1.0	$x^2 + y^2 + z^2 - 1.0^2$
2.0	$x^2 + y^2 + z^2(1.5\cos((z + a/2)/a))^2 - a,\ a = 1.0$



(a) ellipsoid ($t = 0.0$) **(b)** sphere ($t = 1.0$) **(c)** oval ($t = 2.0$)

Fig. 13. Test time-varying volume data.

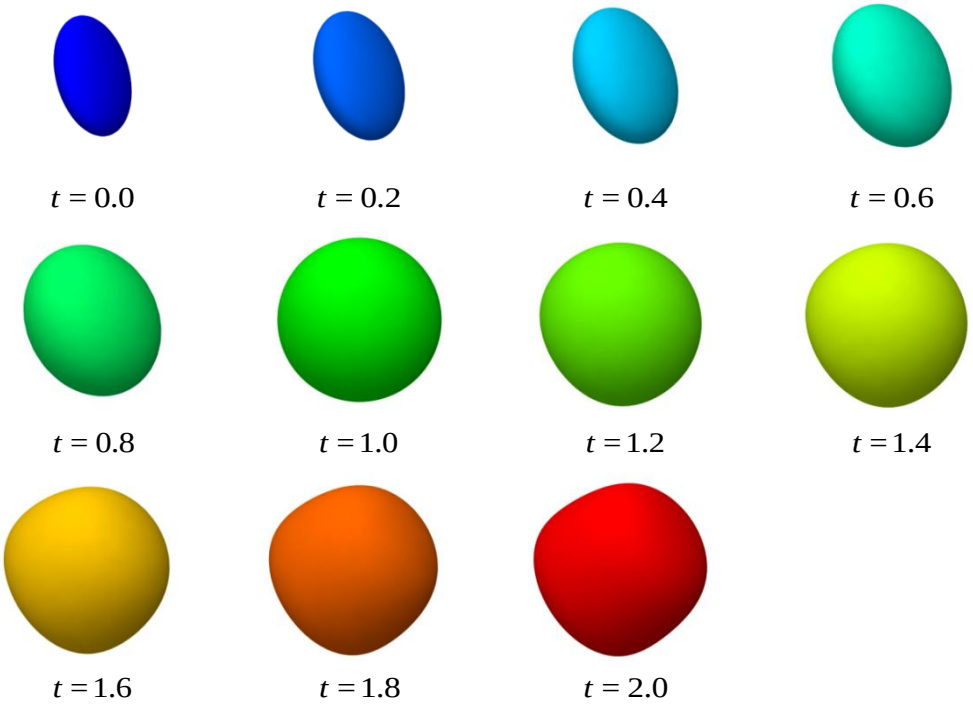


Fig. 14. Reconstruction of time-varying volume data $(\Delta t = 0.2)$.

Table 6. Timing performance for reconstruction of time-varying volume data (64^3) in seconds.

level	1	2	3	4
time	0.08	0.15	0.19	0.28

4.4 Thresholding

We can use a threshold, and in particular a soft threshold, in MSK interpolation, as we can with wavelets. We can discard coefficients below a certain threshold, since many coefficients c_k^j are very small and insignificant. We define a positive threshold $t(j)$ as $t(j) = \mu + \gamma\sigma$, where μ is the average of $|c_k^j|$, σ is the standard deviation of $|c_k^j|$, and γ is a user-controllable parameter. For each j, we choose a threshold $t(j)$ and use $\tilde{c}_k^j$, defined as

$$\tilde{c}_k^j = \begin{cases} c_k^j - t(j), & \text{if } c_k^j \geq t(j) \\ c_k^j + t(j), & \text{if } c_k^j \leq -t(j) \\ 0, & \text{otherwise}, \end{cases}$$

instead of c_k^j in Equation (3). **Table 7** shows the percentage of coefficients remaining after this thresholding process for the squirrel data set. **Fig. 15** shows surfaces reconstructed with thresholding coefficients of about 82% ($\gamma = -1/8$) at a fixed level of detail (level=11). Even though many coefficients have been removed, we can find that the rendered image qualities of reconstruction surfaces are very similar. If the effective encoding scheme that supports fast random access to the remaining coefficients can be implemented, then it would be very useful for various real-time computer graphics applications.

Table 7. Percentage of coefficients remaining after thresholding the squirrel data set (γ is the thresholding parameter).

γ	level						
	1	2	3	4	5	6	7
$-1/8$	27.5	29.1	32.2	35.9	39.1	30.7	26.0
0.0	23.4	25.8	29.1	33.1	35.3	26.6	21.9
1/3	17.6	19.7	23.4	26.9	27.0	19.1	15.5
1/2	15.6	17.5	21.1	24.1	23.6	16.6	13.4
2/3	14.0	15.8	19.0	21.5	20.6	14.5	11.7

(a) no thresholding (117 MB) **(b)** after thresholding (25 MB) where $\gamma = -1/8$

Fig. 15. Removing coefficients by thresholding at a fixed level of detail (level=11).

5. Conclusions and Future Work

We presented a technique to represent multi-dimensional data of implicit functions in a unified form. They allow users to control the level of detail in rendering various types of graphics data, since the generated implicit functions are based on multiscale kernels that provide a robust representation with multiple levels of detail. We showed that our method could effectively be applied to time-varying volumetric data, as well as 3D volume. In addition, the proposed unified data representation can soundly reconstruct surfaces of non-uniformly distributed, noisy or scattered point-sets, and incomplete meshes with holes. Even though pre-processing of our method is slower than that of other surface reconstruction techniques, such as RBFs, but evaluation of function values at arbitrary sample points from the calculated implicit representation is faster. By employing effective techniques such as multi-core programming to enhance the timing performances, the speeds for pre-processing and evaluation increased by about two or threefold. We are currently implementing algorithms to reduce the computational costs of this method, and schemes to encode the coefficients that remain after thresholding effectively, to address various computer graphics applications. We believe that this approach can be applied to fundamental algorithms for mesh deformation, surface compression, edge detection, and scientific visualization of commonly used n-dimensional data sets.

References

[1] J. Carr, R. Beatson, J. Cherrie, T. Mitchell, W. Fright, B. McCallum, and T. Evans, "Reconstruction and Representation of 3D Objects with Radial Basis Functions," in *Proc. of ACM SIGGRAPH 2001*, pp. 67-76, 2001. Article (CrossRef Link)

[2] H. Dinh, G. Turk, and G. Slabaugh, "Reconstructing Surfaces using Anisotropic Basis Functions," in *Proc. of International Conference on Computer Vision 2001*, pp. 606-613, 2001. Article (CrossRef Link)
[3] S. Morse, T. Yoo, D. Chen, P. Rheingans, and K. Subramanian, "Interpolating Implicit Surfaces from Scattered Surface Data using Compactly Supported Radial Basis Functions," in *Proc. of International Conference on Shape Modeling and Applications 2001*, pp. 89-98, 2001. Article (CrossRef Link)
[4] Y. Ohtake, A. Belyaev, and H.-P. Seidel, "A Multi-scale Approach to 3D Scattered Data interpolation with compactly supported basis functions," in *Proc. of Shape Modeling International 2003*, pp. 153-161, 2003. Article (CrossRef Link)
[5] I. Tobor, P. Reuter, and C. Schlick, "Efficient Reconstruction of Large Scattered Geometric Datasets using the Partition of Unity and Radial Basis Functions," in *Proc. of WSCG 2004*, pp. 467-474, 2004. Article (CrossRef Link)
[6] X. Wu, M. T. Wang, and Q. Xia, "Implicit Fitting and Smoothing using Radial Basis Functions with Partition of Unity," in *Proc. of the 9th International Conference on Computer Aided Design and Computer Graphics*, pp. 139-148, 2005.Article (CrossRef Link)
[7] H. Wendland, "Scatted Data Approximation," *Cambridge University Press*, 2005. Article (CrossRef Link)
[8] D. Levin, "The Approximation Power of Moving Least-squares," *Mathematics of Computation*, vol. 67, no. 224, pp. 1517-1531, 1998. Article (CrossRef Link)
[9] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva, "Computing and Rendering Point set Surfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9. No. 1, pp. 3-15, 2003. Article (CrossRef Link)
[10] S. Fleishman, D. Cohen-Or, M. Alexa, and C. Silva, "Progressive Point Set Surfaces," *ACM Transactions on Graphics*, vol. 22, no. 4, pp. 997-1011, 2003. Article (CrossRef Link)
[11] D. Levin, "Geometric Modeling for Scientific Visualization," *Spinger-Verlag*, pp. 37-49. 2003.
[12] S. Fleishman, D. Cohen-Or, and C. Silva, "Robust Moving Least-squares Fitting with Sharp Features," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 544-552, 2005. Article (CrossRef Link)
[13] Y. Lipman, D. Cohen-Or, and D. Leven, "Data-dependent MLS for Faithful Surface Approximation," in Proc. *of the fifth Eurographics symposium on Geometry processing*, pp. 59-67, 2007. Article (CrossRef Link)
[14] J. Manson, G. Petrova, and S. Schaefer, "Streaming Surface Reconstruction using Wavelets," *Computer Graphics Forum*, vol. 27, no. 5, pp. 1411-1420, 2008. Article (CrossRef Link)
[15] C. Walder, B. Schölkopf, and O. Chapelle, "Implicit Surface Modelling with a Globally Regularised Basis of Ccompact Support," *Computer Graphics Forum*, vol. 25, no. 3, pp. 635-644, 2006. Article (CrossRef Link)
[16] R. Schaback and H. Wendland, "Kernel Techniques: From Machine Learning to Meshless Methods," *Acta Numerica*, vol. 15, pp. 543-639, 2006. Article (CrossRef Link)
[17] R. Opfer, "Multiscale Kernels," *Advances in Computational Mathematics*, vol. 25, pp. 357-380, 2006. Article (CrossRef Link)
[18] I. Torbor, P. Reuter, and C. Schlick, "Multi-scale Reconstruction of Implicit Surfaces with Attributes from Large Unorganized Point Sets," in *Proc. of Shape Modeling International*, pp. 19-30, 2004. Article (CrossRef Link)
[19] O. Schenk and K. Gartner, "On Fast Factorization Pivoting Methods for Sparse Symmetric Indefinite Systems," *Elec. Trans. Numer. Anal*, vol. 23, pp. 158-179, 2006.
[20] W. Kim and M. Voss, "Multicore Desktop Programming with Intel Threading Building Blocks," *IEEE Software*, vol. 28, no. 1, pp. 23-31, 2011. Article (CrossRef Link)
[21] L. Buatois, G. Caumon, and B. Levy, "Concurrent Number Cruncher: a GPU Implementation of a General Sparse Linear Solver, *International Journal of Parallel, Emergent and Distributed Systems*, vol. 24, no. 3, pp. 205-223, 2009. Article (CrossRef Link)
[22] D. Alcanta, A. Sharf, F. Abbasinejad, S. Sengupta, M. Mitzenmacher, J. Owens, and N. Amenta, "Real-time Parallel Hashing on the GPU," *ACM Transcations on Graphics*, vol. 28, no. 5, 2009.

Article (CrossRef Link)
[23] N. Nakasato, "Implementation of a Parallel Tree Method on a GPU," *Journal of Computational Science*, 2011. Article (CrossRef Link)

Seongmin Yun received his B.E. and M.E. degree in Multimedia Engineering from Dongguk University, Seoul, Korea, in 2009 and 2011, respectively. He is currently a research associate at Defense Information System Management Group, Korea Institute for Defense Analyses. His research focuses on geometric modeling, real-time rendering, and high performance computing.

Sanghun Park received a B.S. in Mathematics from Sogang University, Seoul, Korea in 1993. He also earned his M.E. and Ph. D. in Computer Science from Sogang University in 1995 and 2000, respectively. He is currently an associate professor at Department of Multimedia, Graduate School of Digital Image and Contents, Dongguk University, Seoul, Korea. His research interests are computer graphics, scientific visualization, and high performance computing. Before joining Dongguk University, he was a research staff member at Computational Visualization Center, Institute for Computational Engineering and Sciences, University of Texas at Austin