Undefined 0 (0) 1
IOS Press

Large Scattered Data Interpolation with Radial Basis Functions and Space Subdivision

Michal Smolik [a,*] and Vaclav Skala [a]

[a] *Department of Computer Science and Engineering, Faculty of Applied Sciences*
University of West Bohemia, Plzen, Czech Republic
E-mail: smolik@kiv.zcu.cz
URL: http://www.VaclavSkala.eu

Abstract. We propose a new approach for the radial basis function (RBF) interpolation of large scattered data sets. It uses the space subdivision technique into independent cells allowing processing of large data sets with low memory requirements and offering high computation speed, together with the possibility of parallel processing as each cell can be processed independently. The proposed RBF interpolation was tested on both synthetic and real data sets. It proved its simplicity, robustness and the ability to handle large data sets together with significant speed-up. In the case of parallel processing, speed-up was experimentally proved when 2 and 4 threads were used.

Keywords: Radial basis functions, interpolation, large data, space subdivision, scattered data

1. Introduction

Interpolation and approximation are probably the most frequent operations used in computational techniques [7]. Several techniques have been developed for data interpolation, but they require some kind of data "ordering", e.g. structured mesh, rectangular mesh, unstructured mesh etc. A typical example is a solution of partial differential equations (PDE), where derivatives are replaced by differences and rectangular or hexagonal meshes are used in the vast majority of cases. However, in many engineering problems, data are not ordered and they are scattered in k-dimensional space, in general. The k-dimensional space is sometimes not only spatial but also contains a time dimension or a dimension relating to age or temperature or other environmental conditions. Usually, in technical applications the scattered data are tessellated using triangulation, but this approach is quite prohibitive for the case of k-dimensional data interpolation because of the computational cost [32].

There exist some techniques using space subdivision to compute a radial basis function (RBF) interpolation. Data point division into sub-domains using an adaptive octree subdivision method and then blending these local functions together with partition of unity is used in [46]. This work is an extension of well-known [31], which uses the multi-level partition of unity to construct surface models from very large sets of points. Spatial down sampling to construct a coarse-to-fine hierarchy of point sets is used in [30]. They interpolate the sets starting from the coarsest level and then they interpolate a point set of the hierarchy, as an offsetting of the interpolating function computed at the previous level. [48] proposed a highly parallel algorithm for RBF interpolation with the time complexity of $O(N)$. The algorithm uses a generalized minimal residual method (GMRES) iterative solver [35] with a restricted additive Schwarz method [5]. The algorithm [6] relies on PetRBF [48]. It improves PetRBF

[*] Corresponding author. E-mail: smolik@kiv.zcu.cz.

Integrated Computer Aided Engineering, IOS Press DRAFT Vol.25, No.1, pp.49-62, ISSN 1069-2509

for surface reconstruction and graphics processing unit (GPU) acceleration. It shows how to make a suitable choice of the algorithm parameters for accurate reconstruction from synthetic, real or incomplete datasets. The algorithm uses domain decomposition to acquire high parallelization. The solution of the original system is built up by solving set of smaller subproblems that interact through their interfaces. [43] optimizes the positions and the weights of the RBF centers and then combines them with a hierarchical domain decomposition technique for the RBF approximation. Other approaches using domain decomposition for the RBF interpolation are [14] which focuses on the parallelization of RBF interpolation with its application for mesh deformation, [2] performs RBF interpolation on divided input points and then iteratively updates all RBF coefficients to create final interpolation, [9] uses multiscale collocation and preconditioners to decrease the condition number of the interpolation matrix, [22] combines the RBF method and the least squares approximation cardinal basis functions (ACBF) preconditioning technique with the domain decomposition method.

All these approaches use space subdivision to compute the RBF interpolation or approximation, but their joining phase is usually not easy to implement. One has many independent interpolations and needs to join them together. These interpolations usually have some overlapping parts and to join them together we need to solve additional systems of equations or iteratively update the resulting interpolation. Our aim is to improve this joining phase and speed-up the calculation of the RBF interpolation as well.

Another approach using virtual points for approximation is used in [25] and [42]. Of course, there are other meshless techniques than RBF, such as discrete smooth interpolation (DSI) [28], which avoids explicitly computing a function defined everywhere and produces values only at the grid points instead. [4], [24] is based on statistical models that include autocorrelation. The scattered data interpolation method described in [20] exploits the topological structure and unsupervised learning algorithm of a $2D$ self-organizing feature map (SOFM) to iteratively create a polygonal surface mesh that takes a general shape of the underlying object. [29] describes a subdivision surface fitting method based on parameter correction to achieve better error measurement. For each given data point, the closest point on the surface is found. This point is expressed as a linear function of the control mesh vertices via basis functions. This function is then defined in a least squares sense as the summation of the squared distances between the data points and the surface points. Another technique, that can be used for meshless interpolation is function-point clustering method (FPCM) [19] which defines a function having a property of being greater in regions where the density of points is higher and being minimal where the density of data points is lower. Multiresolution analysis and wavelets provide useful and efficient tools for representing functions at multiple levels of detail [23]. Multiresolution analysis [8] offers a simple, unified, and theoretically sound approach to deal with the problem of extreme complexity of meshes. The method is based on the approximation of an arbitrary initial mesh by a mesh that has subdivision connectivity and is guaranteed to be within a specified tolerance.

Our goal is to propose a new simple method for interpolation of scattered data points. In many applications, it is necessary to process and interpolate a large amount of data, thus our method has to be able to process such large datasets. There are other interpolation methods, but they are usually quite hard to implement.

Our method is to be easy to implement and it must achieve the same quality of interpolation like other methods. Furthermore, the condition of small memory requirements and low time requirements must be met as well.

2. Radial Basis Functions

Radial basis function (RBF) is a technique for scattered data interpolation [33] and approximation [10], [38]. The RBF interpolation and approximation is computationally more expensive compared to interpolation and approximation methods that use an information about mesh connectivity, because input data are not ordered and there is no known relation between them, i.e. tessellation is not made. Although RBF has a higher computational cost, it can be used for k-dimensional problem solution in many applications, e.g. solution of partial differential equations [21], [49], image reconstruction [44], neural networks [18], [12], [47], fuzzy systems [1], [17], [16], GIS systems [26], optics [34] etc. It should be noted that it does not require any triangulation or tessellation meshing in general. There is no need to know any connectivity of interpolated points, all points are tied up only with distances of each other. Using all these distances we can form the interpolation matrix, which will be shown later.

Integrated Computer Aided Engineering, IOS Press DRAFT Vol.25, No.1, pp.49-62, ISSN 1069-2509

The RBF is a function whose value depends only on the distance from its center point. Due to the use of distance functions, the RBFs can be easily implemented to reconstruct the surface using scattered data in 2D, 3D or higher dimensional spaces. It should be noted that the RBF interpolation is not separable by a dimension.

Radial function interpolants have a helpful property of being invariant under all Euclidean transformations, i.e. translations, rotations and reflections. It does not matter whether we first compute the RBF interpolation function and then apply a Euclidean transformation, or if we first transform all the data and then compute the radial function interpolants. This is a result of the fact that Euclidean transformations are characterized by orthonormal transformation matrices and are therefore two-norm invariant. Radial basis functions can be divided into two groups according to their influence. The first group are "global" RBFs [36], for example:

$$\begin{aligned}
&\text{Thin Plate Spline} && \varphi(r) = r^2 \log r \\
&\text{Gauss function} && \varphi(r) = e^{-(\epsilon r)^2} \\
&\text{Inverse Quadric} && \varphi(r) = \frac{1}{1+(\epsilon r)^2} \\
&\text{Inverse Multiquadric} && \varphi(r) = \frac{1}{\sqrt{1+(\epsilon r)^2}} \\
&\text{Multiquadric} && \varphi(r) = \sqrt{1+(\epsilon r)^2}
\end{aligned} \tag{1}$$

where ϵ is the shape parameter of the radial basis function [11]. Application of global RBFs usually leads to ill-conditioned system, especially in the case of large data sets with a large span [27], [39].

The "local" RBFs were introduced in [45] as compactly supported RBF (CSRBF) and satisfy the following condition:

$$\begin{aligned}
\varphi(r) &= (1-r)_+^q P(r) \\
&= \begin{cases} (1-r)^q P(r) & 0 \le r \le 1 \\ 0 & r > 1 \end{cases}
\end{aligned} \tag{2}$$

where $P(r)$ is a polynomial function and q is a parameter. The subscript in $(1-r)_+^q$ means:

$$(1-r)_+ = \begin{cases} (1-r) & (1-r) \ge 0 \\ 0 & (1-r) < 0 \end{cases} \tag{3}$$

Typical examples of CSRBF are

$$\begin{aligned}
\varphi_1(r) &= (1-\hat{r})_+ \\
\varphi_2(r) &= (1-\hat{r})_+^3(3\hat{r}+1) \\
\varphi_3(r) &= (1-\hat{r})_+^5(8\hat{r}^2+5\hat{r}+1) \\
\varphi_4(r) &= (1-\hat{r})_+^2 \\
\varphi_5(r) &= (1-\hat{r})_+^4(4\hat{r}+1) \\
\varphi_6(r) &= (1-\hat{r})_+^6(35\hat{r}^2+18\hat{r}+3) \\
\varphi_7(r) &= (1-\hat{r})_+^8(32\hat{r}^3+25\hat{r}^2+8\hat{r}+1) \\
\varphi_8(r) &= (1-\hat{r})_+^3 \\
\varphi_9(r) &= (1-\hat{r})_+^3(5\hat{r}+1) \\
\varphi_{10}(r) &= (1-\hat{r})_+^7(16\hat{r}^2+7\hat{r}+1)
\end{aligned} \tag{4}$$

where $\hat{r} = \epsilon r$ and ϵ is the shape parameter of the radial basis function, see Figure 1 for a visualization of Eq. (4).

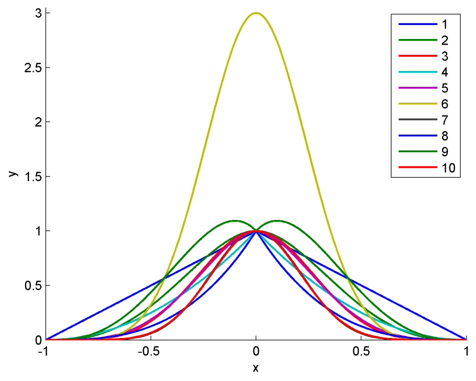


Fig. 1. Examples of CSRBF from Eq. (4).

2.1. *Radial Basis Function interpolation*

RBF interpolation was originally introduced by [15] and is based on computing of the distance of two points in any k-dimensional space. It is defined by the function

$$f(\boldsymbol{x}) = \sum_{j=1}^{M} \lambda_j \varphi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) \tag{5}$$

where λ_j are weights of the RBFs, M is the number of the radial basis functions, i.e. the number of inter-

Integrated Computer Aided Engineering, IOS Press DRAFT Vol.25, No.1, pp.49-62, ISSN 1069-2509

polation points, and φ is the radial basis function. For a given dataset of points with associated values, i.e. in the case of scalar values $\{\boldsymbol{x}_i, h_i\}_1^M$, the following linear system of equations is obtained:

$$h_i = f(\boldsymbol{x}_i) = \sum_{j=1}^{M} \lambda_j \varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|) \quad \text{for } \forall i \in \{1, \dots, M\} \tag{6}$$

where λ_j are weights to be computed; see Figure 2 for a visual interpretation of Eq. (5) or Eq. (6) for a $2\frac{1}{2}D$ function. Point in $2\frac{1}{2}D$ is a $2D$ point associated with a scalar value. The same also applies to $3D$ point associated with a scalar value, thus $3\frac{1}{2}D$ point.

Equation Eq. (6) can be rewritten in a matrix form as

$$\boldsymbol{A}\boldsymbol{\lambda} = \boldsymbol{h}. \tag{7}$$

As $\varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|) = \varphi(\|\boldsymbol{x}_j - \boldsymbol{x}_i\|)$ the matrix $\boldsymbol{A}$ is symmetrical.

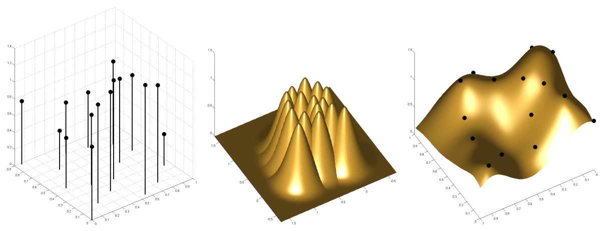


Fig. 2. Data values, the RBF collocation functions, the resulting interpolant.

The RBF interpolation can use "global" or "local" functions. When using "global" radial basis functions, the matrix $\boldsymbol{A}$ will be full, but when using "local" radial basis functions, the matrix $\boldsymbol{A}$ might be sparse, which can be beneficial when solving the system of linear equations $\boldsymbol{A}\boldsymbol{\lambda} = \boldsymbol{h}$.

In the case of the vector data, i.e. $\{\boldsymbol{x}_i, \boldsymbol{h}_i\}_1^M$ values $\boldsymbol{h}_i$ are actually vectors, the RBF is to be performed for each coordinate of the vector $\boldsymbol{h}_i$.

3. Proposed approach

In this section we describe our new proposed approach for large data sets RBF interpolation. The proposed interpolation uses space subdivision to speed-up the computation and to significantly reduce high memory requirements [26], [25]. The algorithm consists of three main steps. The first one is the space subdivision, the second one is the RBF interpolation and the last one is the joining procedure of interpolated cells ("blending") to create the final interpolation. The pseudo-code of the proposed approach is in Algorithm 1 and 2. We show the speed-up of the proposed algorithm compared to the standard one for RBF interpolation as well.

Algorithm 1 Pseudocode of the proposed RBF interpolation method.

```
1: procedure RBF(Points P)                 ▷ $P_i = \{\mathbf{x}_i, h_i\}$
2:     for all cells in grid do
3:         Enlarge cell by $1/\epsilon$   ▷ where $\epsilon$ is the shape
4:                                              parameter
5:         $p \leftarrow$ Points in enlarged cell
6:         Compute RBF interpolation of $p$
```

Algorithm 2 Pseudocode of interpolated value calculation using the proposed RBF interpolation method.

```
1: procedure RBF(Point p)                  ▷ $p = \{x, y\}$
2:     Find neighboring cells
3:     Compute distances to cells
4:     Compute interpolated RBF values for all cells
5:     Blend RBF values together    ▷ using distances
6:                                         to cells
```

3.1. Space subdivision

The approach proposed is based on a divide and conquer (D&C) strategy, and therefore input data set is split into several subsets. In our case, we will use a rectangular grid of the size $n \times m$ domains for $2\frac{1}{2}D$ input data, resp. $n \times m \times l$ domains for $3\frac{1}{2}D$ input data. The grid does not have to be necessarily regular and we can adjust it according to the properties of the input data set. We will use an orthogonal regular grid of domains for simplicity of explanation of the proposed approach.

The input points need to be divided into some cells according to the created grid for the space subdivision. Every domain of the grid needs to be enlarged to a cell and contains a few more points from the neighborhood, see Figure 3. We will present the reason for this later in this paper.

The input points need to be divided into some number of cells. This number can be estimated according

Integrated Computer Aided Engineering, IOS Press DRAFT Vol.25, No.1, pp.49-62, ISSN 1069-2509

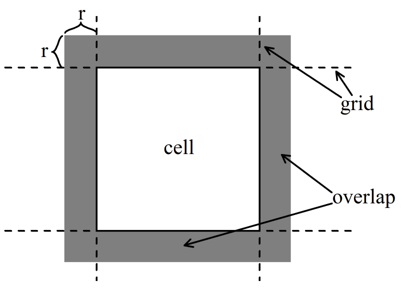


Fig. 3. $2D$ regular orthogonal grid with one cell visualized. Each cell contains points from the grid domain plus points from the overlapping parts with neighborhood domains.

to the memory available. The RBF interpolation matrix for n points in the cell has the size $n \times n$ elements, which are usually stored as double precision numbers. The size of the matrix in bytes is given as

$$size = sizeof(double) \cdot n^2 = 8n^2 \ (Byte). \quad (8)$$

Using this formula we can easily find out the maximal average number of points in cells, see Figure 4, and set up easily the size of the grid needed for the subdivision.

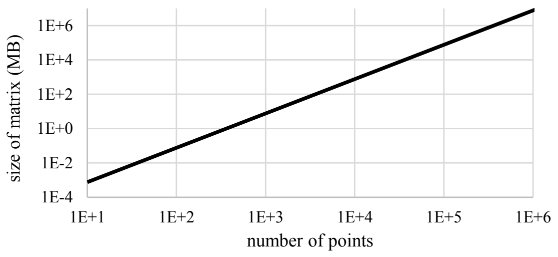


Fig. 4. The size of the RBF interpolation matrix for different number of interpolated points. The matrix is stored in double precision and its size is in MB if full matrix structure is used. The memory requirements are $O(N^2)$, where N is the number of points.

Data points are generally scattered, so it might be further possible in the extreme case that nearly all points lie within one cell. In this case it would be necessary to split this cell again. Another possible case is when no point lie inside a cell. In this case, the shape parameter and grid size for RBF interpolation is inappropriately selected and must be changed in the sense that the influencing of the basis function is greater and sufficient for the data interpolation [27].

3.2. *Cells RBF interpolation*

Now, we have all input points divided into overlapping cells and thus can do the RBF interpolation. Radial basis functions have one parameter, which is the shape parameter ϵ. In the proposed approach, we use the "local" radial basis functions (CSRBFs), as they have the restricted maximal distance for the influence of the RBF interpolation. The shape parameter should be chosen so, that $\frac{1}{\epsilon}$ is equal to the size r of the overlapping of each domain (Figure 3), resp. vice versa. Points on the border of a cell are exactly r away from the grid domain and RBF center points with a larger distance than r will not have any influence on the interpolated value inside a domain of the grid.

Points inside a cell need to be interpolated using the RBF interpolation with CSRBF. This interpolation is done using the standard calculation of the linear system of equations Eq. (6). Each cell is interpolated as an independent cell and thus the calculation can be done totally in parallel. This parallel calculation will increase the performance and speed-up the RBF computation for each cell. The only problem that can arise is the memory consumption, as we need to store multiple interpolation matrices at once, so this should be kept in mind when computing the size of a grid for space subdivision.

For each cell we get one set of weighting values of the RBF interpolation $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_n]^T$. These values have to be stored for later use. The matrix used for their calculation, i.e. the RBF interpolation matrix, can be discarded.

3.3. *Blending of cells and reconstruction function*

The interpolated cells computed in the previous step are overlapping each other. In this section, we show how to join, i.e. blend, them together to create a final continuous interpolation function that covers all the cells and thus all the input points for the interpolation as well.

The total width of overlapping parts is $2r$. To blend all the neighborhood cells together, we will do some kind of bilinear interpolation ("blending") between them. The computed value from each cell needs to be multiplied with a coefficient α. The coefficients α_i are computed as

$$\alpha' = min(1, \frac{distance\ from\ the\ border}{2r}), \quad (9)$$

where $distance\ from\ the\ border$ is the shortest distance from the location to the border and it is calculated using the Euclidean metric. However, for the axes-aligned grid, the distance can be calculated using Chebychev metric, which is defined as

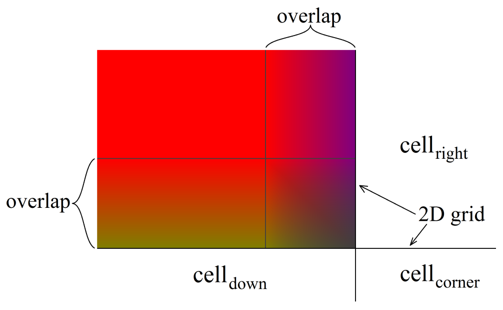
$$distance(P, Q) = \max_i(|p_i - q_i|), \tag{10}$$

where $P = [p_1, ..., p_k]^T$ and $Q = [q_1, ..., q_k]^T$ are two points in k-dimensional space.

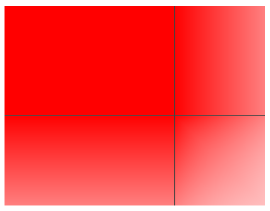
The final coefficients α_i are computed using Eq. (9) as

$$\alpha_i = \frac{\alpha_i{}'}{\sum_{j=1}^{2^k} \alpha_j{}'}, \tag{11}$$

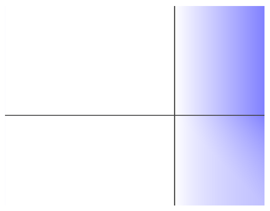
where k is the dimension, i.e. $k = 2$ for $2\frac{1}{2}D$ or $k = 3$ for $3\frac{1}{2}D$ input data. The visual representation of coefficients is shown in Figure 5.



(a)



(b) Red component of the color from the Figure 5a.



(c) Blue component of the color from the Figure 5a.

Fig. 5. Bilinear interpolation between cells for the overlapped areas. Red part of color represents the coefficient for the main cell value, green part of color represents the coefficient for the down cell value and blue part of color represents the coefficient for the right cell value. The value for the corner cell is calculated as $1 - (red + green + blue)$.

Knowing all the coefficients α_i and all function values from the RBF interpolations of cells, we can compute the final value of the proposed radial basis function interpolation algorithm for large scattered data interpolation .

$$f(\boldsymbol{x}) = \sum_{i=1}^{2^k} \alpha_i \left(\sum_{j=1}^{M_i} \lambda_j \varphi \left(\left\| \boldsymbol{x} - \boldsymbol{x}_j^{(i)} \right\| \right) \right), \tag{12}$$

where k is the dimension, i.e. $k = 2$ or $k = 3$, M_i is the number of points in the i-th cell, α_i is the coefficient from Eq. (11) and $\boldsymbol{x}_j^{(i)}$ are interpolation points in a cell.

During the blending phase we perform the interpolation between the interpolations of cells. The result of the blending phase is thus again the interpolation of all input points, as the resulting function passes through all input points.

3.4. Speed-up of the proposed approach (interpolation)

The proposed approach uses space subdivision to speed-up the calculation of radial basis function interpolation and to reduce the needed memory as well. In the following, we will use the notation shown in Figure 6.

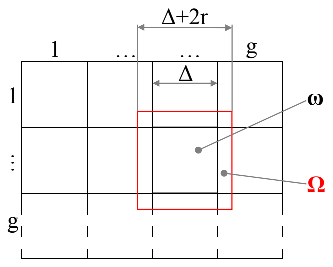


Fig. 6. Visualization of a grid.

The value g is equal to the number of divisions in each dimension, k is the dimension, Δ is the size of one domain, r is the size of the overlap for each cell and is equal to the radius of the RBF.

The number of points n in the area ω can be estimated in the case of uniform distribution as

$$n = \frac{N}{g^k}, \tag{13}$$

where N is the total number of points for the interpolation and g is equal to the number of divisions in each dimension. Every domain ω is enlarged by the overlap

r, see Figure 3, at every side of the domain; thus the enlargement of the domain is equal to

$$\xi = \frac{\Delta + 2r}{\Delta} = 1 + \frac{2r}{\Delta}. \tag{14}$$

The average number of points in the enlarged cell Ω is equal to

$$m = \frac{N}{g^k}\xi^k. \tag{15}$$

When computing the RBF interpolation, we need to solve a system of linear equations (LSE). Let us assume that solving an LSE of size $N \times N$ has the time complexity $O(N^3)$. The time complexity of our proposed interpolation for one enlarged cell ω, i.e. the cell Ω, is

$$O\left(\left(\frac{N}{g^k}\xi^k\right)^3\right). \tag{16}$$

Therefore, the expected speed-up of the proposed algorithm compared to the standard one is

$$\nu = \frac{O\left(N^3\right)}{O\left(g^k\left(\frac{N}{g^k}\xi^k\right)^3\right)} = O\left(\frac{N^3}{g^k\left(\frac{N}{g^k}\xi^k\right)^3}\right) = O\left(\left(\frac{g^2}{\xi^3}\right)^k\right), \tag{17}$$

where $\nu \gg 1$ for the most grid resolutions, as can be seen in Figure 7, which was generated for $\Delta = 1$ and the overlap $r = 0.2$, i.e. 20% overlap at each side of every domain. It should be noted, that the axis for ν is in logarithmic scaling.

The time complexity of our proposed approach for the RBF interpolation is

$$O\left(g^k\left(\frac{N}{g^k}\xi^k\right)^3\right) = O\left(\frac{N}{n}\left(n\xi^k\right)^3\right), \tag{18}$$

where n and ξ can be constants. Then the only variable in Eq. (18) is N. Thus, the time complexity of the proposed approach is $O\left(N\right)$, but only in cases when the data points are uniformly distributed. Otherwise the worst time complexity of the proposed approach is $O\left(N^3\right)$.

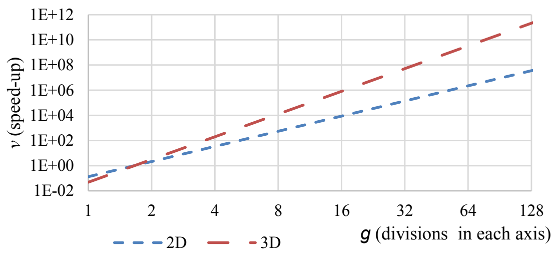


Fig. 7. Expected speed-up of the proposed algorithm according to Eq. (17) for different numbers g, i.e. resolution of the grid, for $\Delta = 1$ and the overlap $r = 0.2$.

3.5. *Speed-up of the proposed approach (function evaluation)*

The proposed approach does not speed-up only the RBF interpolation calculation, but it also speed-up the evaluation of the interpolation function as well. The time complexity of the function evaluation for the standard RBF is

$$O\left(N\right). \tag{19}$$

The time complexity of the function evaluation for our proposed approach for the RBF interpolation is

$$O\left(2^k\frac{N}{g^k}\xi^k\right). \tag{20}$$

Using Eqs. (19) and (20), we can compute the speed-up of our proposed algorithm when computing one function value of the RBF interpolation:

$$\eta = \frac{O\left(N\right)}{O\left(2^k\frac{N}{g^k}\xi^k\right)} = O\left(\frac{N}{2^k\frac{N}{g^k}\xi^k}\right) = O\left(\left(\frac{g}{2\xi}\right)^k\right). \tag{21}$$

For most grid resolutions the speed-up $\eta \gg 1$, as can be seen in Figure 8, which was generated for $\Delta = 1$ and the overlap $r = 0.2$, i.e. 20% overlap at each side of every domain. It should be noted that the axis for η is in logarithmic scaling.

4. Results

In this section we show the results of our proposed approach. This approach for RBF interpolation is espe-

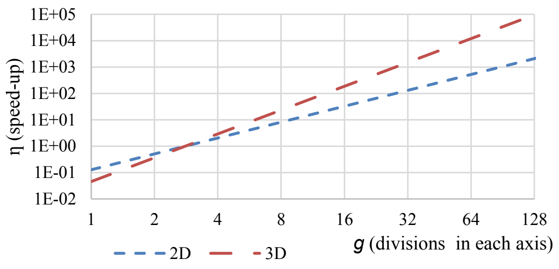


Fig. 8. Expected speed-up of function evaluation using the proposed algorithm according to Eq. (21) for different numbers g, i.e. resolution of the grid, for $\Delta = 1$ and the overlap $r = 0.2$.

cially convenient for large data set interpolation. However, in the first sub-section we test it for the case of its simplicity only with small synthetically generated data sets to show some basic results of the proposed method for RBF interpolation.

In the second sub-section we tested our approach with real data sets. The second example is a data set containing more than $6 \cdot 10^6$ points, which is much more than the standard RBF interpolation is able to handle and compute on an ordinary computer.

Any of the CSRBFs in Eq. (4) can be used for the proposed RBF interpolation. However, in the tests we present results for one basis function, namely

$$\varphi_5(r) = (1 - \epsilon r)^4_+ (4\epsilon r + 1). \tag{22}$$

We tested the proposed approach also with global radial basis functions, specifically with thin plate spline (TPS) and Gauss function. The results for global RBFs are very similar to those when using CSRBFs.

The implementation of the RBF interpolation was performed in MATLAB and tested on a PC with the following configuration:

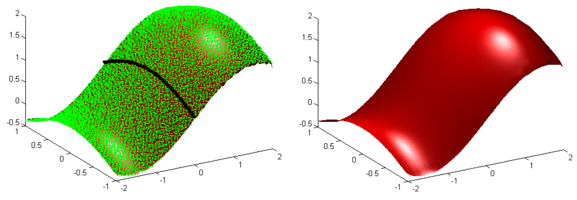
- CPU: Intel® Core™ i7-920 (4 × 2.67 GHz + hyper-threading),
- memory: 22GB RAM,
- operation system: Microsoft Windows 8 64 bit.

4.1. $2\frac{1}{2}$D synthetic data

We first tested the proposed RBF interpolation on a synthetic data set of points using the function

$$f(x, y) = \sin(x) + \cos(y). \tag{23}$$

We sampled the function at 10^4 random positions with a Halton distribution (A.1 in [10]) where $x \in [-2; 2]$ and $y \in [-1; 1]$, see Figure 9a. We used a grid of the size 2×1 and $\epsilon = 5$ and 10% of overlapping. The result of the proposed interpolation can be seen in Figure 9b. The result is continuous.



(a) Input points (b) RBF interpolated surface

Fig. 9. 10^4 input points were used to test the proposed RBF interpolation.

We measured the difference of function values of the two RBF interpolations of two cells on their common border before the blending phase, see Figure 9a. We should note that for this test, we did not blend these two RBF interpolations. The absolute difference between those two cells along the border is visualized in Figure 10.

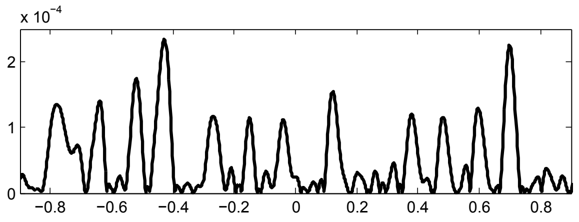


Fig. 10. Absolute difference in function values along the common border between the RBF interpolations of two cells without the blending phase, i.e. without the linear interpolation between cells.

We measured the difference of function values between each cell RBF interpolation and the original function Eq. (23) on the common border before the blending phase. The difference between each cell and the original function is visualized in Figure 11. We should note that for this test, we did not blend these two interpolations in any way.

Two cells are interpolated using RBF interpolation independently and then blend together. We measured the interpolation error between blended cells and the original function Eq. (23). The results are visualized in Figure 12. The proposed interpolation is continuous, without any disparity between domains.

We measured the interpolation error between the proposed RBF interpolation and the original function

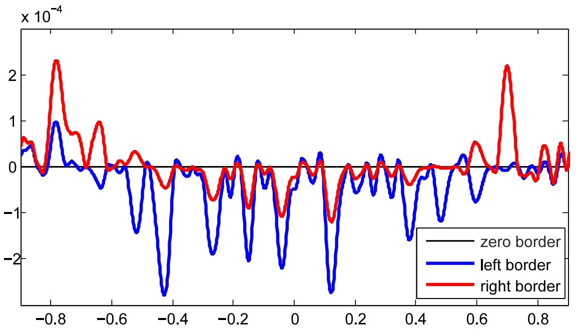


Fig. 11. Difference of function values along the common border between the interpolation of each cell without the blending phase, i.e. without the linear interpolation between cells, and the original function Eq. (23).

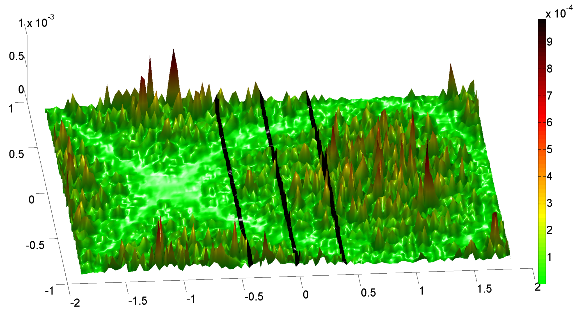


Fig. 12. Difference in function values between the proposed RBF interpolation and the original function Eq. (23).

Eq. (23) on the common border. The error is visualized in Figure 13. It can be seen that the error has a behavior similar to that represented in Figure 12.

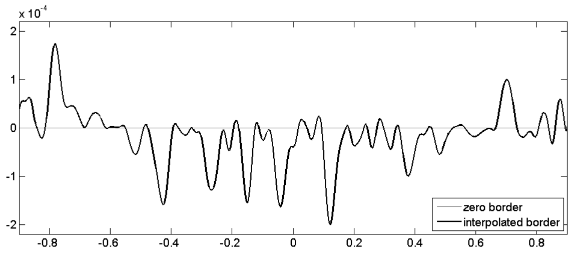


Fig. 13. Difference in function value between the proposed RBF interpolation with blending phase and the original function Eq. (23).

The same measurement as in Figure 10 was done for a different percentage of cells overlapping, see Figure 14. It can be seen that the error decreases and for 100% overlapping this error is 0, as both the RBF interpolations use all points for the interpolation of their cell. It means that the proposed RBF interpolation is continuous, i.e. waterproof.

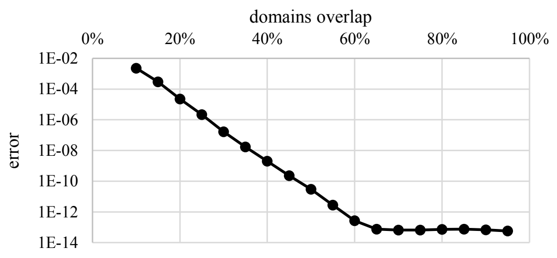


Fig. 14. Absolute difference in function values along the common border between interpolations of the two cells for different sizes of overlapping parts (for 100% $error = 0$).

However, we need also to measure the quality of this RBF interpolation. For this purpose we compare our proposed method using the space subdivision with the standard RBF interpolation method (2.2 in [10]) using $2 \cdot 10^4$ randomly sampled points with the uniform distribution of the function [10]:

$$\begin{aligned} f(x,y) &= 3(1-x)^2 e^{(-x^2-(y+1)^2)} \\ &\quad - 10(\frac{x}{5} - x^3 - y^5) e^{(-x^2-y^2)} \\ &\quad - \frac{1}{3} e^{(-(x+1)^2-y^2)}, \end{aligned} \tag{24}$$

where $x \in [-3;3]$ and $y \in [-3;3]$.

We used a grid of size 4×4 and the shape parameter with the size 20% of the domain edge length. The result of this interpolation is presented in Figure 15. The standard RBF interpolation used the same points, the same basis function and the same shape parameter for interpolation.

To evaluate the quality of the interpolation we generated $1.5 \cdot 10^5$ randomly sampled points with Halton distribution where $x \in [-3;3]$ and $y \in [-3;3]$. Then we computed function values of both the interpolations and evaluate the absolute error of each interpolation. For each point $P_i = [x_i, y_i]^T$ we compute absolute error

$$Err_i = \|RBF(P_i) - f(x_i, y_i)\|_2\,, \tag{25}$$

where $RBF(P_i)$ is the interpolated value at point P_i using standard RBF interpolation on the whole dataset and RBF interpolation of the proposed approach, $f(x_i, y_i)$ is the function value of Eq. (24). The

Integrated Computer Aided Engineering, IOS Press DRAFT Vol.25, No.1, pp.49-62, ISSN 1069-2509

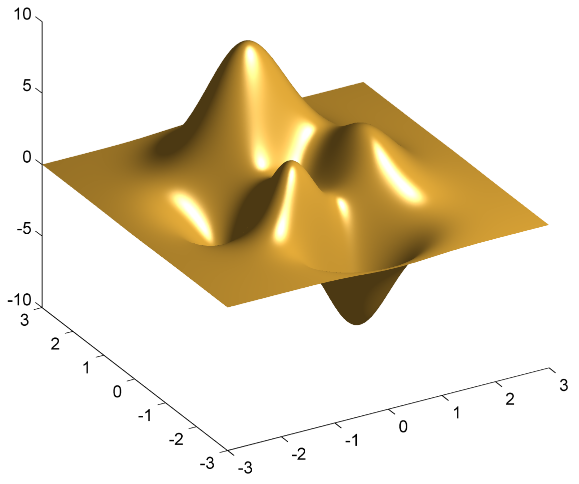


Fig. 15. The result of RBF interpolation using the proposed method with space subdivision.

Figure 16 presents distribution histograms of the interpolation errors.

As both histograms in Figure 16 are visually identical, we created a difference histogram between the two histograms. In Figure 17 it can be seen that the interpolation errors distribution is almost identical. The difference in both histograms differs only slightly, see Figure 17. Thus both interpolations have almost the same quality.

We also computed the average interpolation error for each RBF interpolation. The result is in Table 1. We can see that both average interpolation errors are almost the same, there is only a difference of 0.03%. Knowing all results from quality measurements we can say that our proposed RBF interpolation has almost identical quality as the standard RBF interpolation.

Table 1

Average interpolation error of the proposed approach and the standard RBF interpolation. The interpolation error difference between both measured methods is only 0.03%.

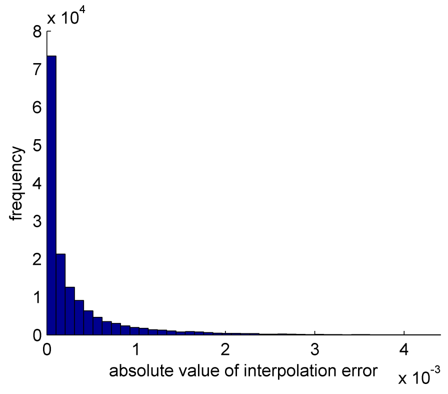
	proposed approach	standard RBF interpolation
mean absolute error	$3.1371 \cdot 10^{-4}$	$3.1362 \cdot 10^{-4}$

4.2. Real data set

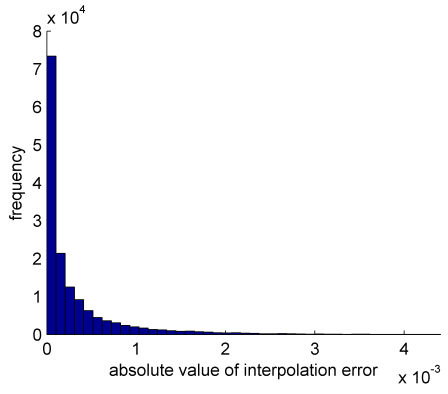
The proposed approach is mainly suited for large data interpolation. For this reason we chose to use a real data set. The LiDAR data of Mount Saint Helens[1] in Skamania County, Washington, contains scanned height data. The data set consists of $6,743,176$ $2D$ points with associated heights, i.e. $2\frac{1}{2}D$ data.

We chose to divide the input data set into a regular grid in a way such that the inside of a domain is going to be on average $5,000$ points. To make square domains, we created a grid of the size 29×46, as the data range is around $2.1 \cdot 10^4$ $ft \times 3.3 \cdot 10^4$ ft, i.e. $6.4 \cdot 10^3$ $m \times 1.0 \cdot 10^4$ m, in x and y coordinates. The visualization of the created grid domains is in Figure 18.

To perform the RBF interpolation, we needed to choose the shape parameter ϵ of the CSRBF. We tested different values of the shape parameter and selected the best shape parameter which has the size of 20% of the domain edge length. Each cell will therefore contain



(a) Standard RBF interpolation.



(b) Proposed method RBF interpolation.

Fig. 16. Histograms of interpolation errors.

[1]http://www.liblas.org/samples/

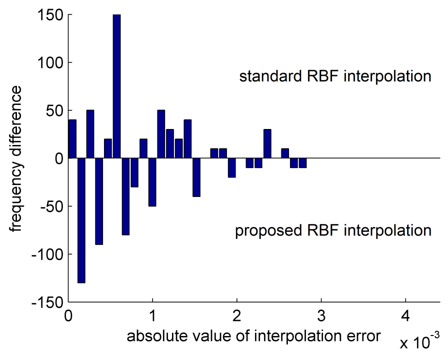


Fig. 17. Difference of histograms in Figure 16. Positive values mean that the standard RBF interpolation has more errors with the specific absolute value of interpolation error and the negative values mean the same for the proposed method for RBF interpolation.

approximately

$$5,000 \cdot (1 + 2 \cdot 0.2)^2 = 9,800 \qquad (26)$$

points. The number of points inside the cell is almost double times more than number of points inside the domain, but the final speed-up will still be very high. For clarity we can estimate the speed-up of the proposed algorithm compared to the standard one as:

$$speed\text{-}up = \frac{6,743,176^3}{29 \cdot 46 \cdot 9,800^3} \approx 2.4 \cdot 10^5. \qquad (27)$$

It can be seen that the speed-up is significant and we save a lot of calculations. The expected speed-up of the function evaluation is

$$speed\text{-}up = \frac{6,743,176}{2^2 \cdot 9,800} \approx 172. \qquad (28)$$

This means that each RBF function computation for a given $\boldsymbol{x}$ is approximately 172 times faster.

Moreover, the standard algorithm for RBF interpolation would require around 330 TB to save the full interpolation matrix to the memory when double precision is used.

The data set divided into cells was interpolated one cell after another. We used one RBF interpolated cell to reconstruct the terrain inside one domain of the grid without blending step. The result can be seen together with the grid of domains in Figure 18.

Figure 19 presents the result of the proposed RBF interpolation method. We used Eq. (12) to compute interpolation of the height values of the terrain for the

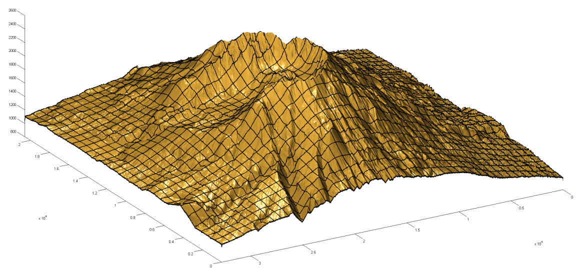


Fig. 18. Visualization of the interpolated terrain produced only as a visualization of each domain separately. The orthogonal grid used for the space subdivision with resolution of 29 × 46 is visualized on the terrain as well.

visualization. This terrain does not have any discontinuity because of the proposed blending procedure.

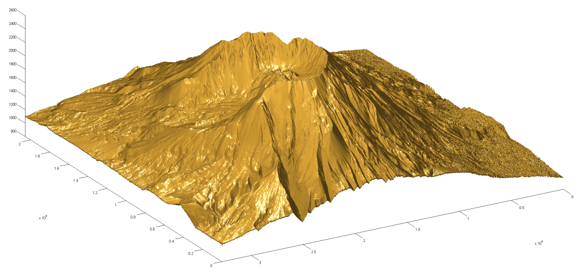


Fig. 19. Visualization of the final result of the proposed method for large scattered data interpolation with the RBF and space subdivision.

The proposed algorithm can be easily parallelized as the RBF interpolation of each cell of the grid can be done separately and thus in parallel. We measured the running time of the interpolation when using 1 or 2 or 4 threads. The resulting speed-up in MATLAB is in the Table 2. It can be seen that the speed-up is high because the threads do not have to wait for any synchronization and are independent of each other.

Table 2

Parallel speed-up of the proposed method compared to the serial version of this method.

number of threads	1	2	4
speed-up	1	1.791	3.172

We tested our proposed approach with another data set too. We chose a model of the terrain[2] which con-

[2]http://www.badking.com.au/site/shop/environment/mountain-terrain/

tains $131,044$ points with associated heights, i.e. $2\frac{1}{2}D$ data.

We divided the input data set to a regular grid so that a domain contains $3,000$ points in average. We created a grid of the size 6×6, with the data range is around $0.2172\ miles \times 0.2172\ miles$ in x and y coordinates. The visualization of the created grid of domains is in Figure 20.

For the shape parameter, we used the size of 20% of the domain edge length. Therefore, each cell contains around

$$3,000 \cdot (1 + 2 \cdot 0.2)^2 = 5,880 \tag{29}$$

points. It is almost double times more, but the final speed-up will still be very high. We can estimate the speed-up of the proposed algorithm compared to the standard one:

$$speed\text{-}up = \frac{131,044^3}{6 \cdot 6 \cdot 5,880^3} \approx 3 \cdot 10^2. \tag{30}$$

It can be seen that the speed-up is significant and will save us a lot of calculations. The speed-up of interpolating function evaluation is

$$speed\text{-}up = \frac{131,044}{2^2 \cdot 5,880} \approx 5.6. \tag{31}$$

Moreover, the standard algorithm for the RBF interpolation needs around 128 GB to save the full interpolation matrix to the memory when double precision is used.

The data set divided into cells was interpolated one cell after another. We did a visualization of this RBF interpolation without doing any blending procedure. We used one RBF interpolated cell to reconstruct the terrain inside each domain of the grid. The result can be seen in Figure 20, together with a visualization of the grid.

Figure 21 presents the result of the proposed RBF interpolation method. We used Eq. (12) to compute the height values of the terrain for the visualization. This terrain is continuous and does not have any discontinuity because of the proposed blending procedure.

If CSRBF is used, many elements in the interpolation matrix are equal to zero, as the matrix is sparse in general. To decrease the memory requirements and be able to solve large interpolation matrices we can use a sparse matrix data structure. There are several existing sparse matrix representations. e.g. [3], [37], [26]. The main difference among existing storage formats is the sparsity pattern, or the structure of nonzero elements, for they are best suited. In our implementation, the coordinate format is used, which is briefly described in the following.

The coordinate (COO) format [13] is the simplest storage scheme. The sparse matrix is represented by three arrays: $data$, where the nonzero values are stored, row, where the row index of each nonzero element is kept, and col, where the column indices of the nonzero values are stored. The benefit of this format is its generality, i.e. an arbitrary sparse matrix can be represented by the COO format and the required storage is always proportional to the number of nonzero values. The disadvantage of the COO format is that both row and column indices are stored explicitly, which reduces the efficiency of memory transactions (e.g. read operations).

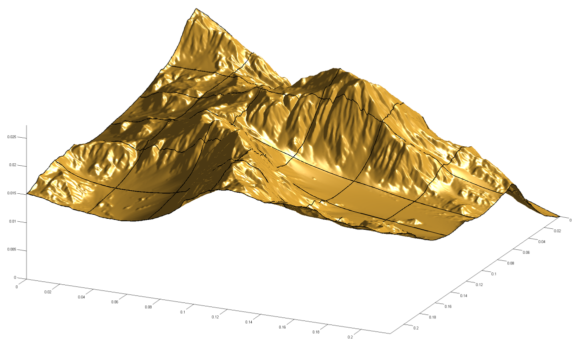


Fig. 20. Visualization of the interpolated terrain produced only as a visualization of each domain separately. The orthogonal grid used for the space subdivision with resolution of 6×6 is visualized on the terrain as well.

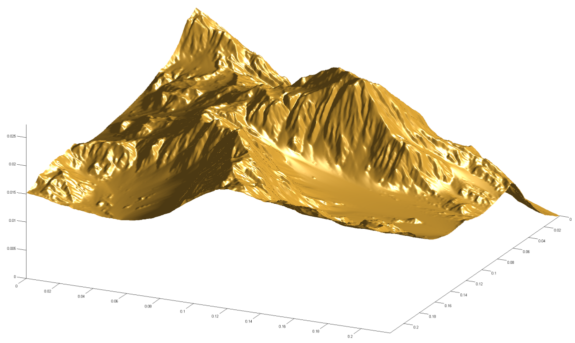


Fig. 21. Visualization of the final result of the proposed method for large scattered data interpolation with the RBF and space subdivision.

Integrated Computer Aided Engineering, IOS Press DRAFT Vol.25, No.1, pp.49-62, ISSN 1069-2509

Moreover, note that the elements in the interpolation matrix are zero for far away points, when CSRBFs are used. Therefore, we do not need to compute the elements for all pairs of points, so the kd-tree (A.2 in [10]) is used for computing the interpolation matrix.

As the proposed approach also needs to be compared with the standard one for RBF interpolation; we used the dataset in Figure 21 which contains $131,044$ points for interpolation. For the shape parameter for RBF interpolation we used the size of $1/30$ of the data range. We measured the running times of our algorithm running in sequential version for different grid resolutions and computed the speed-up compared to the standard algorithm for RBF interpolation, see Table 3. Both methods are using sparse matrix with COO format and kd-tree structure. We also measured the memory requirements and the results are in Table 4.

Table 3

Speed-up of the proposed approach for large scattered data interpoloation compared to the standard RBF interpolation. Both methods are using sparse matrix with COO format and kd-tree structure.

grid resolution	4×4	6×6	8×8	10×10	12×12
speed-up	1.69	1.83	2.06	2.28	2.57

According to the results in Table 3, the proposed algorithm is faster than the standard one and the speed-up is increasing with increasing of the grid resolution; both methods used the COO sparse matrix structure. We could not compute the speed-up when using the full matrix data structure as we were unable to fit such large data into the available memory for the standard algorithm.

Table 4

Memory requirements for our proposed method and for the standard RBF interpolation method. The proposed method was tested with full matrix data structure and also using sparse matrix with COO format together with kd-tree structure. The standard algorithm for RBF interpolation uses sparse matrix with COO format together with kd-tree structure.

	proposed method		standard method
grid size	kd-tree and sparse matrix	full matrix	kd-tree and sparse matrix
4x4	590 MB	6,900 MB	8,800 MB
6x6	290 MB	2,300 MB	
8x8	180 MB	1,000 MB	
10x10	125 MB	500 MB	
12x12	95 MB	400 MB	

According to the results in Table 4, the proposed approach has much lower memory requirements than the standard one. Therefore our approach enables to compute the RBF interpolation for very large datasets even on computers with standard memory size.

5. Conclusion

We presented a new approach for radial basis function interpolation of scattered data. It computes the interpolation on partly overlapping cells and then blends these interpolations together to create the final interpolation of the whole data set. This approach is especially efficient for large scattered data interpolation, as it reduces the memory required. It significantly speeds up computation of the interpolated value. The proposed approach is suitable for parallelization and it was tested on synthetic and large real data sets. It proved its robustness and high performance.

In future the proposed approach will be used for vector fields interpolation of large data sets based on [40], [41] and considering also vector field characteristics. We plan to modify the proposed method for $3D$ scattered data interpolation and approximation. In the case of $3D$, point data will be split into overlapping cubes according to the grid. The joining phase, i.e. blending, will be almost the same as when blending $2D$ cells.

Acknowledgment

The authors would like to thank their colleagues at the University of West Bohemia, Plzen, for their discussions and suggestions, especially to Zuzana Majdisova, and anonymous reviewers for their valuable comments and hints provided. The research was supported by projects Czech Science Foundation (GACR) No. 17-05534S and SGS 2016-013.

References

[1] H. Adeli and A. Karim, Fuzzy-wavelet RBFNN model for freeway incident detection, *Journal of Transportation Engineering* **126**(6) (2000), 464–471.

[2] R.K. Beatson, W.A. Light and S.D. Billings, Fast Solution of the Radial Basis Function Interpolation Equations: Domain Decomposition Methods, *SIAM J. Scientific Computing* **22**(5) (2001), 1717–1740. doi:10.1137/S1064827599361771.

[3] N. Bell and M. Garland, Implementing sparse matrix-vector multiplication on throughput-oriented processors, in: *Proceedings of the conference on high performance computing networking, storage and analysis*, ACM, 2009, pp. 1–11.

[4] T.Q. Bui, T.N. Nguyen and H. Nguyen-Dang, A moving Kriging interpolation-based meshless method for numerical simulation of Kirchhoff plate problems, *International Journal for Numerical Methods in Engineering* **77**(10) (2009), 1371–1395.

[5] X.-C. Cai and M. Sarkis, A restricted additive Schwarz preconditioner for general sparse linear systems, *SIAM Journal on scientific computing* **21**(2) (1999), 792–797.

[6] S. Cuomo, A. Galletti, G. Giunta and A. Starace, Surface reconstruction from scattered point via RBF interpolation on GPU, in: *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, IEEE, 2013, pp. 433–440.

[7] P.J. Davis, *Interpolation and approximation*, Courier Corporation, 1975. ISBN ISBN 978-0-486-62495-2.

[8] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery and W. Stuetzle, Multiresolution analysis of arbitrary meshes, in: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM, 1995, pp. 173–182.

[9] P. Farrell and J. Pestana, Block preconditioners for linear systems arising from multiscale collocation with compactly supported RBFs, *Numerical Lin. Alg. with Applic.* **22**(4) (2015), 731–747. doi:10.1002/nla.1984.

[10] G.E. Fasshauer, *Meshfree approximation methods with MATLAB*, Vol. 6, World Scientific, 2007.

[11] B. Fornberg and C. Piret, On choosing a radial basis function and a shape parameter when solving a convective PDE on a sphere, *J. Comput. Physics* **227**(5) (2008), 2758–2780. doi:10.1016/j.jcp.2007.11.016.

[12] S. Ghosh-Dastidar, H. Adeli and N. Dadmehr, Principal component analysis-enhanced cosine radial basis function neural network for robust epilepsy and seizure detection, *IEEE Transactions on Biomedical Engineering* **55**(2) (2008), 512–518.

[13] J.R. Gilbert, C. Moler and R. Schreiber, Sparse matrices in MATLAB: Design and implementation, *SIAM Journal on Matrix Analysis and Applications* **13**(1) (1992), 333–356.

[14] G. Haase, D. Martin and G. Offner, Towards RBF Interpolation on Heterogeneous HPC Systems, in: *Large-Scale Scientific Computing - 10th International Conference, LSSC 2015*, 2015, pp. 182–190.

[15] R.L. Hardy, Multiquadric equations of topography and other irregular surfaces, *Journal of geophysical research* **76**(8) (1971), 1905–1915.

[16] C.-F. Hsu, C.-M. Lin and R.-G. Yeh, Supervisory adaptive dynamic RBF-based neural-fuzzy control system design for unknown nonlinear systems, *Applied Soft Computing* **13**(4) (2013), 1620–1626.

[17] A. Karim and H. Adeli, Comparison of fuzzy-wavelet radial basis function neural network freeway incident detection model with California algorithm, *Journal of Transportation Engineering* **128**(1) (2002), 21–30.

[18] A. Karim and H. Adeli, Radial basis function neural network for work zone capacity and queue estimation, *Journal of Transportation Engineering* **129**(5) (2003), 494–503.

[19] J.O. Katz and F.J. Rohlf, Function-point cluster analysis, *Systematic Biology* **22**(3) (1973), 295–301.

[20] G.K. Knopf and A. Sangole, Interpolating scattered data using 2D self-organizing feature maps, *Graphical Models* **66**(1) (2004), 50–69.

[21] E. Larsson and B. Fornberg, A numerical study of some radial basis function based solution methods for elliptic PDEs, *Computers & Mathematics with Applications* **46**(5) (2003), 891–902.

[22] L. Ling and E.J. Kansa, Preconditioning for radial basis functions with domain decomposition methods, *Mathematical and Computer Modelling* **40**(13) (2004), 1413–1427. doi:10.1016/j.mcm.2005.01.002.

[23] M. Lounsbery, T.D. DeRose and J. Warren, Multiresolution analysis for surfaces of arbitrary topological type, *ACM Transactions on Graphics (TOG)* **16**(1) (1997), 34–73.

[24] Y.Z. Ma, J.J. Royer, H. Wang, Y. Wang and T. Zhang, Factorial kriging for multiscale modelling, *Journal of the Southern African Institute of Mining and Metallurgy* **114**(8) (2014), 651–659.

[25] Z. Majdisova and V. Skala, A Radial Basis Function Approximation for Large Datasets, in: *Proceedings of SIGRAD 2016*, Linköping University Electronic Press, 2016, pp. 9–14.

[26] Z. Majdisova and V. Skala, Big geo data surface approximation using radial basis functions: A comparative study, *Computers & Geosciences* **109** (2017a), 51–58.

[27] Z. Majdisova and V. Skala, Radial Basis Function Approximations: Comparison and Applications, *Applied Mathematical Modelling* **51** (2017b), 728–743.

[28] J.-L. Mallet, Discrete smooth interpolation, *ACM Transactions on Graphics (TOG)* **8**(2) (1989), 121–144.

[29] M. Marinov and L. Kobbelt, Optimization methods for scattered data approximation with subdivision surfaces, *Graphical Models* **67**(5) (2005), 452–473.

[30] Y. Ohtake, A.G. Belyaev and H. Seidel, A Multi-scale Approach to 3D Scattered Data Interpolation with Compactly Supported Basis Function, in: *2003 International Conference on Shape Modeling and Applications (SMI 2003)*, 2003, pp. 153–164292. doi:10.1109/SMI.2003.1199611.

[31] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk and H.-P. Seidel, Multi-level partition of unity implicits, in: *ACM Siggraph 2005 Courses*, ACM, 2005, p. 173.

[32] J. O'Rourke, A.J. Mallinckrodt et al., Computational geometry in C, *Computers in Physics* **9**(1) (1995), 55–55.

[33] R. Pan and V. Skala, A two-level approach to implicit surface modeling with compactly supported radial basis functions, *Engineering with Computers* **27**(3) (2011), 299–307.

[34] G. Prakash, M. Kulkarni and U. Sripati, Using RBF Neural Networks and Kullback-Leibler distance to classify channel models in Free Space Optics, in: *Optical Engineering (ICOE), 2012 International Conference on*, IEEE, 2012, pp. 1–6.

[35] Y. Saad and M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM Journal on scientific and statistical computing* **7**(3) (1986), 856–869.

[36] I.P. Schagen, Interpolation in two dimensions - a new technique, *IMA Journal of Applied Mathematics* **23**(1) (1979), 53–59.

[37] I. Simecek, Sparse matrix computations using the quadtree storage format, in: *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2009 11th International Symposium on*, IEEE, 2009, pp. 168–173.

[38] V. Skala, Meshless Interpolations for Computer Graphics, Visualization and Games, in: *Eurographics 2015 - Tutorials*, 2015. doi:10.2312/egt.20151046.

[39] V. Skala, RBF Interpolation with CSRBF of Large Data Sets, *Procedia Computer Science* **108** (2017), 2433–2437.

[40] M. Smolik and V. Skala, Vector Field Interpolation with Radial Basis Functions, in: *Proceedings of SIGRAD 2016*, Linköping University Electronic Press, 2016, pp. 15–21.

[41] M. Smolik and V. Skala, Classification of Critical Points Using a Second Order Derivative, *Procedia Computer Science* **108** (2017), 2373–2377.

[42] M. Smolik, V. Skala and O. Nedved, A Comparative Study of LOWESS and RBF Approximations for Visualization, in: *Computational Science and Its Applications - ICCSA 2016 - 16th International Conference, Part II*, 2016, pp. 405–419. doi:10.1007/978-3-319-42108-7_31.

[43] J. Süßmuth, Q. Meyer and G. Greiner, Surface reconstruction based on hierarchical floating radial basis functions, *Computer Graphics Forum* **29**(6) (2010), 1854–1864.

[44] K. Uhlir and V. Skala, Reconstruction of damaged images using radial basis functions, in: *Signal Processing Conference, 2005 13th European*, IEEE, 2005, pp. 1–4.

[45] H. Wendland, Computational aspects of radial basis function approximation, *Studies in Computational Mathematics* **12** (2006), 231–256.

[46] J. Yang, Z. Wang, C. Zhu and Q. Peng, Implicit Surface Reconstruction with Radial Basis Functions, in: *International Conference on Computer Vision and Computer Graphics*, Springer, 2007, pp. 5–12.

[47] L. Yingwei, N. Sundararajan and P. Saratchandran, Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm, *IEEE Transactions on neural networks* **9**(2) (1998), 308–318.

[48] R. Yokota, L.A. Barba and M.G. Knepley, PetRBF-A parallel O(N) algorithm for radial basis function interpolation with Gaussians, *Computer Methods in Applied Mechanics and Engineering* **199**(25) (2010), 1793–1804.

[49] X. Zhang, K.Z. Song, M.W. Lu and X. Liu, Meshless methods based on collocation with radial basis functions, *Computational mechanics* **26**(4) (2000), 333–343.