Fast Interpolation and Approximation of Scattered Multidimensional and Dynamic Data Using Radial Basis Functions

VACLAV SKALA
Department of Computer Science and Engineering
University of West Bohemia, Faculty of Applied Sciences
Univerzitni 8, CZ 306 14 Plzen
Czech Republic
http://www.VaclavSkala.eu

Abstract: Interpolation or approximation of scattered data is very often task in engineering problems. The Radial Basis Functions (RBF) interpolation is convenient for scattered (un-ordered) data sets in k-dimensional space, in general. This approach is convenient especially for a higher dimension $k > 2$ as the conversion to an ordered data set, e.g. using tessellation, is computationally very expensive. The RBF interpolation is not separable and it is based on distance of two points. It leads to a solution of a Linear System of Equations (LSE) $\boldsymbol{Ax} = \boldsymbol{b}$. There are two main groups of interpolating functions: 'global" and "local". Application of "local" functions, called Compactly Supporting RBF (CSFBF), can significantly decrease computational cost as they lead to a system of linear equations with a sparse matrix.

In this paper the RBF interpolation theory is briefly introduced at the "application level" including some basic principles and computational issues and an incremental RBF computation is presented and approximation RBF as well.

The RBF interpolation or approximation can be used also for image reconstruction, inpainting removal, for solution of Partial Differential Equations (PDE), in GIS systems, digital elevation model DEM etc.

Key-Words: - Radial basis function, RBF interpolation, image reconstruction, incremental computation, RBF approximation, fast matrix multiplication

1 Introduction

Interpolation and approximation are probably the most frequent operations used in computational techniques. Several techniques have been developed for data interpolation, but they expect some kind of data "ordering", e.g. structured mesh, rectangular mesh, unstructured mesh etc. The 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. 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.

An interesting technique is k-dimensional data interpolation using Radial Basis Functions (RBF). The RBF interpolation is computationally more expensive because interpolated data are not ordered, but offers quite interesting applications at acceptable computational cost, e.g. solution of partial differential equations, image reconstruction, neural networks, fuzzy systems, GIS systems, optics and interferometry etc.

2 Problem Formulation

Interpolation is very often used and mostly linear interpolation is used in technical applications. Let us analyze first different types of data to be processed. Also there is a question whether the Euclidean space representation is the best for computing and engineering applications. It is well known that the division operation is very dangerous in numerical computations and causes severe problems in numerical methods. Also it is known that computations can be made in the projective extension of the Euclidean space [20] [21] [23] [27]. The projective formulation of numerical problems leads to very interesting questions, e.g. an explicit solution of LSE is equivalent to the cross-product, like why the division operation in the Gauss-Seidel or similar methods is needed [21]? The projective space representation and the principle of duality also help to solve some problems more efficiently [19] [20] [25]. Also Non-rational uniform B-Splines

(NURBS) are actually curves or surfaces defined using the projective extension of the Euclidean space.

In the following we use the Euclidean space representation to explain the fundamental principles and we will explore incremental computation of RBF interpolation and approximation, as well.

3 Data Classification

Before analyzing methods for interpolation, it is reasonable to classify data to be processed. It seems to be simple, but let us explore it more deeply. Generally, the data can be represented by:

1. Coordinates, e.g. by points $\{\boldsymbol{x}_i\}_1^M$ in computer graphics, which forms triangular mesh in E^2 or represent a surface of an object in E^3.
2. Coordinates and associated values $\{\langle \boldsymbol{x}_i, \boldsymbol{h}_i \rangle\}_1^M$, e.g. coordinates of points $\boldsymbol{x}_i$ associated with vector values $\boldsymbol{h}_i$ with each point or associated with scalar values (potential field), e.g. representing temperatures etc..

The dimensionality of a vector of coordinates $dim(\boldsymbol{x}_i) = \mathrm{k}$, i.e. $\boldsymbol{x}_i = [x_1, \dots, x_k]^T$, while the dimensionality of a vector of values $dim(\boldsymbol{h}_i) = p$, i.e. $\boldsymbol{h}_i = [h_1, \dots, h_p]^T$.

It can be seen that those two cases are quite different cases if an interpolation is to be used. Also data can be:

- hierarchical
- non-hierarchical

or

- adaptive to some physical phenomena
- non-adaptive

and

- static
- dynamic (t-variant) in coordinates $\boldsymbol{x}_i$ or in values $\boldsymbol{h}_i$ or both!

In a selection of an interpolation technique we have to respect if they are “ordered” or “un-ordered” as well. Then the data sets can be classified as follows.

• Un-ordered	- Scattered - Clustered
• Ordered	- Unstructured - Structured - Non-regular - Semi-regular - Regular

Table 1: A simple classification of data

In the case of un-ordered data, mostly some tessellation techniques like triangulation in the E^2 case or tetrahedronization in the E^3 case are used and generally an unstructured mesh is obtained.

The semi-regular mesh is obtained just in the case when data are ordered in a rectangular grid and Delaunay triangulation is used. It should be noted that this is a very unstable situation, as due to some small shifts in coordinates, the tessellation can be totally changed.

Interpolation techniques on “ordered” data sets are well known and used in many packages.

Let us explore how to interpolate values $\boldsymbol{h}_i$ in the given un-ordered $\{\langle \boldsymbol{x}_i, \boldsymbol{h}_i \rangle\}_1^M$ data set. Of course, there is a theoretical possibility to use a tessellation in order to get an ordered unstructured mesh, but this process is computationally very expensive as the computational complexity of the tessellation grows with the dimension k non-linearly and complexity of the implementation grows as well.

On the other hand, there are interpolation techniques applicable for un-ordered data sets. One of such technique is based on Radial Basis Functions (RBF) which is especially convenient for the interpolation in the k-dimensional space, especially if $k > 2$. The RBF interpolation based on radial basis functions is quite simple from a mathematical point of view.

For an explanation of the RBF interpolation let us consider the case, when h_i are scalar values. The RBF interpolation is based on computing of the distance of two points in the k -dimensional space and is defined by a function [2], [3]

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

It means that for the given data set $\{\langle \boldsymbol{x}_i, h_i \rangle\}_1^M$, where h_i are associated values to be interpolated and $\boldsymbol{x}_i$ are domain coordinates. We obtain a linear system of equations

$$h_i = f(\boldsymbol{x}_i) = \sum_{j=1}^{N} \lambda_j \, \varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|) \qquad i = 1, \dots, M$$

where: λ_j are weights to be computed. Due to some stability issues, usually a polynomial $P_k(\boldsymbol{x})$ of a degree k is added to the form, i.e.

$$h_i = f(\boldsymbol{x}_i) = \sum_{j=1}^{M} \lambda_j \, \varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|) + P_k(\boldsymbol{x}_i)$$
$$i = 1, \dots, M$$

For a practical use a linear polynomial

$$P_1(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{x} + a_0$$

in many applications. So the RBF interpolation function has the form:

$$f(\boldsymbol{x}_i) = \sum_{j=1}^{M} \lambda_j\, \varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|) + \boldsymbol{a}^T\boldsymbol{x}_i + a_0$$

$$= \sum_{j=1}^{M} \lambda_j\, \varphi_{i,j} + \boldsymbol{a}^T\boldsymbol{x}_i + a_0$$

where

$$h_i = f(\boldsymbol{x}_i) \qquad i = 1, \dots, M$$

and additional conditions are applied:

$$\sum_{j=1}^{M} \lambda_i = 0 \qquad \sum_{j=1}^{M} \lambda_i \boldsymbol{x}_i = \boldsymbol{0}$$

It can be seen that for k-dimensional case a system of $(M + k + 1)$ LSE has to be solved, where M is a number of points in the dataset and k is the dimensionality of data.

For $k=2$ vectors $\boldsymbol{x}_i$ and $\boldsymbol{a}$ are given as $\boldsymbol{x}_i = [x_i, y_i]^T$ and $\boldsymbol{a} = [a_x, a_y]^T$. Using the matrix notation we can write for 2-dimensions:

$$\begin{bmatrix} \varphi_{1,1} & .. & \varphi_{1,M} & x_1 & y_1 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \varphi_{M,1} & .. & \varphi_{M,M} & x_M & y_M & 1 \\ x_1 & .. & x_M & 0 & 0 & 0 \\ y_1 & .. & y_M & 0 & 0 & 0 \\ 1 & .. & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_M \\ a_x \\ a_y \\ a_0 \end{bmatrix} = \begin{bmatrix} h_1 \\ \vdots \\ h_M \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} \boldsymbol{B} & \boldsymbol{P} \\ \boldsymbol{P}^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix} \qquad \boldsymbol{Ax} = \boldsymbol{b}$$

$$\boldsymbol{a}^T\, \boldsymbol{x}_i + a_0 = a_x\, x_i + a_y\, y_i + a_0$$

It can be seen that for the two-dimensional case and M points given a system of $(M + 3)$ linear equations has to be solved. If "global" functions, e.g. TPS ($\varphi(r) = r^2 lg\, r$), are used the matrix $\boldsymbol{B}$ is "full", if "local" functions (Compactly supported RBF – CSRBF) are used, the matrix $\boldsymbol{B}$ can be sparse.

The radial basis functions interpolation was originally introduced by [5] by introduction of multiquadric method in 1971, which he called Radial Basis Function (RBF) method. Since then many different RFB interpolation schemes have been developed with some specific properties, e.g. Thin-Plate Spline function $\varphi(r) = r^2 lg\, r$, which is called TPS [4], a function $\varphi(r) = e^{-(\epsilon r)^2}$ was proposed by [9] and [12] introduced Compactly Supported RBF (CSRBF) as

$$\varphi(r) = \begin{cases} (1-r)^q\, P(r), & 0 \le r \le 1 \\ 0, & r > 1 \end{cases},$$

where: $P(r)$ is a polynomial function and q is a parameter.

Theoretical problems with stability and solvability were solved by [6] and [13]. Generally, there are two main groups of the RBFs:

- "global" – a typical example is TPS function
- "local" – Compactly supported RBF (CSRBF)

If the "global" functions are taken, the matrix $\boldsymbol{A}$ of the LSE is full and for large M is becoming ill conditioned and problems with convergence can be expected.

On the other hand if the CSRBFs are taken, the matrix $\boldsymbol{A}$ is becoming relatively sparse, i.e. computation of the LSE will be faster, but we need to carefully select the scaling factor and the final function tends to be "blobby" shaped.

"Global" functions	$\phi(r)$
Thin-Plate Spline (TPS)	$r^2 \log r$
Gauss function	$\exp\left(-(\varepsilon r)^2\right)$
Inverse Quadric (IQ)	$1/\left(1+(\varepsilon r)^2\right)$
Inverse multiquadric (IMQ)	$1/\sqrt{1+(\varepsilon r)^2}$
Multiquadric (MQ)	$\sqrt{1+(\varepsilon r)^2}$

Table 1 Typical examples of "global" functions"

ID	**Function**
1	$(1-r)_+$
2	$(1-r)_+^3(3r+1)$
3	$(1-r)_+^5(8r^2+5r+1)$
4	$(1-r)_+^2$
5	$(1-r)_+^4(4r+1)$
6	$(1-r)_+^6(35r^2+18r+3)$
7	$(1-r)_+^8(32r^3+25r^2+8r+1)$
8	$(1-r)_+^3$
9	$(1-r)_+^3(5r+1)$
10	$(1-r)_+^7(16r^2+7r+1)$

Table 2 Typical examples of "local" functions - CSRBF [13]

Tab.2 presents typical examples of CSRBFs. They are defined for the interval (0, 1), but for the practical use a scaling is used, i.e. the value r is multiplied by a scaling factor α, where $0<\alpha<1$. Fig.1

shows behavior of selected CSRBF; on the x axis is a radius value r (negative part is just for illustration of the symmetry properties).

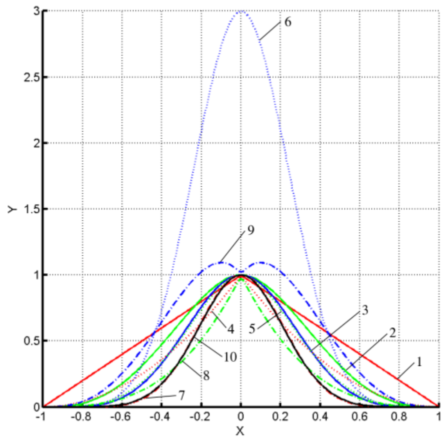

Fig.1 Geometrical properties of CSRBF [13]

4 Matrix Inversion and Multiplication

Matrix multiplication, inversion and solution of a linear system of equations (LSE) are probably the most frequent operations used in computations. RBF interpolation leads naturally to a system of linear equations to be solved. However in the case of "global" RBF the matrix is full, large and ill conditioned. In the case of scalar and static values an iterative methods can be used to obtain a solution. Nevertheless for a multidimensional data represented by

$$\boldsymbol{h} = [h_1, \dots, h_k]^T$$

for static data or for dynamic data, i.e.

$$\boldsymbol{h}(t) = [h_1(t), \dots, h_k(t)]^T$$

increment methods cannot be used and the system of linear system of equations representing RBF interpolation has to be solved by an inverse matrix computation due to time and computational stability.

Let us consider some operations with block matrices again (we assume that all operations are correct and matrices are non-singular in general etc.). The matrix inversion is defined as follows:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}^{-1} = \begin{bmatrix} (\boldsymbol{A}-\boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D}-\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1} \\ -(\boldsymbol{D}-\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{C}\boldsymbol{A}^{-1} & (\boldsymbol{D}-\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1} \end{bmatrix}$$

Let us consider a matrix $\boldsymbol{M}$ of $(n+m)\times(n+m)$ and a matrix $\boldsymbol{A}$ of $n\times n$ in the following block form:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^T & \boldsymbol{D} \end{bmatrix}$$

Then the inverse of the matrix $\boldsymbol{M}$ applying the rule above can be written as:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^T & \boldsymbol{D} \end{bmatrix}^{-1} = \begin{bmatrix} (\boldsymbol{A}-\boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{B}^T)^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D}-\boldsymbol{B}^T\boldsymbol{A}^{-1}\boldsymbol{B})^{-1} \\ -(\boldsymbol{D}-\boldsymbol{B}^T\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{A}^{-1} & (\boldsymbol{D}-\boldsymbol{B}^T\boldsymbol{A}^{-1}\boldsymbol{B})^{-1} \end{bmatrix}$$

where the matrix $\boldsymbol{A}^{-1}$ is known and matrices $\boldsymbol{A}$, $\boldsymbol{A}^{-1}$, $\boldsymbol{D}$ and $\boldsymbol{D}^{-1}$ are symmetrical and semi-positive definite.

Computation complexities are as follow:

$\boldsymbol{Q} = \boldsymbol{A}^{-1}\boldsymbol{B}$	$O(mn^2)$
$\boldsymbol{T} = \boldsymbol{B}^T\boldsymbol{Q}$ $= \boldsymbol{B}^T\boldsymbol{A}^{-1}\boldsymbol{B}$	$O(m^2n)$
$\boldsymbol{D}^{-1}$	$O(m^3)$
$\boldsymbol{R} = \boldsymbol{D}^{-1}\boldsymbol{B}^T$	$O(m^2n)$
$\boldsymbol{W} = \boldsymbol{B}\boldsymbol{R}$	$O(mn^2)$
$\boldsymbol{Z} = \boldsymbol{D} - \boldsymbol{T}$	$O(m^2)$

Table 3 : Computational complexity

By definition and symmetry

$$\boldsymbol{T} = \boldsymbol{T}^T \qquad \boldsymbol{Q}^T = \boldsymbol{B}^T\boldsymbol{A}^{-1} \quad \boldsymbol{Z} = \boldsymbol{Z}^T$$

We can further simplify the matrix inversion. Then

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}^{-1} = \begin{bmatrix} (\boldsymbol{A}-\boldsymbol{B}\boldsymbol{R})^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D}-\boldsymbol{B}^T\boldsymbol{Q})^{-1} \\ -(\boldsymbol{D}-\boldsymbol{B}^T\boldsymbol{Q})^{-1}\boldsymbol{B}^T\boldsymbol{A}^{-1} & (\boldsymbol{D}-\boldsymbol{B}^T\boldsymbol{Q})^{-1} \end{bmatrix}$$
$$= \begin{bmatrix} (\boldsymbol{A}-\boldsymbol{W})^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D}-\boldsymbol{T})^{-1} \\ -(\boldsymbol{D}-\boldsymbol{T})^{-1}\boldsymbol{Q}^T & (\boldsymbol{D}-\boldsymbol{T})^{-1} \end{bmatrix}$$
$$= \begin{bmatrix} (\boldsymbol{A}-\boldsymbol{W})^{-1} & -\boldsymbol{Q}\,\boldsymbol{Z}^{-1} \\ -\boldsymbol{Z}^{-1}\boldsymbol{Q}^T & \boldsymbol{Z}^{-1} \end{bmatrix}$$

Finally we get

$$\boldsymbol{M}^{-1} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}^{-1} = \begin{bmatrix} (\boldsymbol{A}-\boldsymbol{W})^{-1} & -\boldsymbol{Q}\,\boldsymbol{Z}^{-1} \\ -(\boldsymbol{Q}\,\boldsymbol{Z}^{-1})^T & \boldsymbol{Z}^{-1} \end{bmatrix}$$

In the worst case of $n = m$.

$\boldsymbol{Z}^{-1}$	$O(m^3)$
$(\boldsymbol{A}-\boldsymbol{W})^{-1}$	$O(n^3)$
$\boldsymbol{Q}\,\boldsymbol{Z}^{-1}$	$O(m^2n)$

Table 4: Computational complexity

It means that we have a formula to solve a system of linear equations for the special case when the matrix is symmetrical. According to the RBF interpolation definition above, the matrix $\boldsymbol{A}$ defines RBF interpolation of $n-3$ points in the E^2 case and

$n-4$ points in the E^3 case. Other matrices express the m points added to the interpolation.

As the inversion and matrix multiplication operations are $O(N^3)$ complexity then for $m=n$ the complexity of the matrix $\boldsymbol{M}^{-1}$ computation is $O(N^3)$, while sub-matrix operations are of $O((\frac{N}{2})^3)$ complexity, i.e. the **TOTAL cost expected is** *O((2n)³) = 8 O(n³)*, if the inverse of $\boldsymbol{M}$ is computed and the computation will be slightly faster and inversion operation will be slightly more stable as well.

From the efficiency point of view, **the worst case** is for $m=n$. For the case of $m \neq n$ the efficiency of computation will be higher. Total computational complexity for $m \neq n$ given as

No of operation with the given complexity	
$O(mn^2)$	2
$O(m^2n)$	2
$O(m^3)$	2
$O(n^3)$	1
$O(n^2)$	1

Table 5: For $m=1$, i.e. for one point insertion, the complexity is $O(n^2)$ only.

It can be seen that there are the following critical operations:

Matrix storing – as we expect to process many points, i.e. number of points

$$n \in < 10^3, 10^6 >,$$

and memory requirements grow with $O(n^2)$ complexity, i.e. memory consumption will be

$$O_{mem} \in < 10^6, 10^{12} >$$

which is becoming prohibitive also from the stability issue. As the matrices $\boldsymbol{M}$, $\boldsymbol{D}$, $\boldsymbol{A}$ are symmetrical we can save approx. ½ of memory requirements.

Matrix multiplication – this operation seems to be simple as the standard formula

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

$i = 1, \dots, n \;\; j-1, \dots, p$ is used. However this operation is of $O(n^3)$ in general. There are some more effective algorithms for special cases, e.g. Strassen's algorithm [30], [W2] with computational complexity $O(n^{2.81})$, but matrices must be of $2^k \times 2^k$ sizes, or Coopersmith-Winograd's algorithm [29] with $O(n^{2.38})$ complexity. However it should be noted that algorithms are based on recursion, time for memory allocation for matrices and for data transmissions are not considered.

It should be noted, that ***umerical precision due to many matrix operations is weakly stable*** [W3].

Experiments made recently showed that those factors significantly decrease the efficiency of the Strassen's algorithm.

The Strassen's algorithm for matrices multiplication is actually based on the observation that

$$bc + ad = (a+b)(c+d) - ac - bd$$

so instead of having 4 multiplications we need only 3 multiplications as follows (it is expected that all operations are valid). Let us consider

$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \times \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix}$$

Then

$$\begin{aligned} \boldsymbol{M}_1 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{22}) \times (\boldsymbol{B}_{11} + \boldsymbol{B}_{22}) \\ \boldsymbol{M}_2 &= (\boldsymbol{A}_{21} + \boldsymbol{A}_{22}) \times \boldsymbol{B}_{11} \\ \boldsymbol{M}_3 &= \boldsymbol{A}_{11} \times (\boldsymbol{B}_{12} - \boldsymbol{B}_{22}) \\ \boldsymbol{M}_4 &= \boldsymbol{A}_{22} \times (\boldsymbol{B}_{21} - \boldsymbol{B}_{11}) \\ \boldsymbol{M}_5 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{12}) \times \boldsymbol{B}_{22} \\ \boldsymbol{M}_6 &= (\boldsymbol{A}_{21} - \boldsymbol{A}_{11}) \times (\boldsymbol{B}_{11} + \boldsymbol{B}_{12}) \\ \boldsymbol{M}_7 &= (\boldsymbol{A}_{12} - \boldsymbol{A}_{22}) \times (\boldsymbol{B}_{21} + \boldsymbol{B}_{22}) \end{aligned}$$

Then the final matrix $\boldsymbol{C}$ is given as

$$\begin{aligned} \boldsymbol{C}_{11} &= \boldsymbol{M}_1 + \boldsymbol{M}_4 - \boldsymbol{M}_5 + \boldsymbol{M}_7 \\ \boldsymbol{C}_{12} &= \boldsymbol{M}_3 + \boldsymbol{M}_5 \\ \boldsymbol{C}_{21} &= \boldsymbol{M}_2 + \boldsymbol{M}_4 \\ \boldsymbol{C}_{22} &= \boldsymbol{M}_1 - \boldsymbol{M}_2 + \boldsymbol{M}_3 + \boldsymbol{M}_6 \end{aligned}$$

If the above rules are applied recursively we get algorithm complexity $O(n^{log_2 7}) \approx O(n^{2.81})$. It should be noted that the matrices must be of $2^k \times 2^k$ size and there is a lot of memory allocation and data transmission from/to a matrix.

Let us more explore our case of the RFB interpolation, when the **RBF matrix is symmetrical** and the inverse matrix is symmetrical as well. It means that we do not need to store n^2 elements, but only $n(n-1)/2$, which is significantly lower memory requirements. It should be noted that a result of multiplication of two symmetrical matrices is not generally a symmetrical matrix.

```
procedure MULT (in: A symmetrical, B; out: C);
# Matrix A: symmetrical; B: general; C: general
# Sizes are to be legal
# Initialization
$c_{ij}$ := 0;        $c_{ij}$ := $a_{ii}$*$b_{ij}$;        # for all i,j
```

```
for i := 1, n-1
  for j := i+1, n
    for k := i+1, n
    {       c_ij +:= a_ik*b_kj;
            c_ki +:= a_ik*b_kj
            #      Coherence of caching
            #      q := b_kj  ;
            #      c_ij +:= a_ik*q;
            #      c_ki +:= a_ik*q
    }
```

Algorithm 1

As an element a_{ij} in a matrix is stored in a linear data structure as b_q, i.e. in a vector, a new mapping function, if the matrix $\boldsymbol{A}$ is stored "row by row" compressed for, as

$$\begin{aligned} q = (i-1)n + j - i + 1 &= in - n + j - i + 1 \\ &= i(n-1) - (n-1)j \\ &= (i-j)(n-1) \quad \text{for} \quad i < j \end{aligned}$$

Now, the formula for the matrix multiplication for a symmetric matrix $\boldsymbol{A}$ has to be modified, see Alg.1.

It can be seen that the computational complexity is $2\,(n-1)\frac{1}{2}n\frac{1}{2}n = \frac{1}{2}n^2(n-1)$ which is again $O(n^3)$, but computation will be faster about four times.

Matrix inversion is generally of $O(n^3)$, i.e. $\frac{1}{6}n(n+1)(n+2) - \frac{1}{2}n(n+1)$, complexity, if explicit solution is made. The explicit solution is necessary in the case of multidimensional or dynamic (t-variant) data interpolation.

5 Incremental computation

As for many applications, the number of points is high and some data are to be deleted and new inserted,.

It is not possible to recompute the whole LSE due to computational complexity. In this case the incremental computation of RBF is to be used. The algorithm itself is simple [28] and can be simply described as follows:

Incremental RBF computation
The main question to be answered is:

Is it possible to use already computed RFB interpolation if a new point is to be included into the data set?

If the answer is positive it should lead to significant decrease of computational complexity.

In the following we will present how a new point can be inserted, a selected point can be removed and also how to select the best candidate for a removal according to an error caused by this point removal.

Let us consider some operations with block matrices (we will assume that all operations are correct and matrices are non-singular in general etc.).

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}^{-1} = \begin{bmatrix} (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1} & -\boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1} \\ -(\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{C}\boldsymbol{A}^{-1} & (\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1} \end{bmatrix}$$

Let us consider a matrix $\boldsymbol{M}$ of *(n+1)×(n+1)* and a matrix $\boldsymbol{A}$ of $n \times n$ in the following block form:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^T & c \end{bmatrix}$$

Then the inverse of the matrix $\boldsymbol{M}$ applying the rule above can be written as:

$$\boldsymbol{M}^{-1} = \begin{bmatrix} \left(\boldsymbol{A} - \frac{1}{c}\boldsymbol{b}\boldsymbol{b}^T\right)^{-1} & -\frac{1}{k}\boldsymbol{A}^{-1}\boldsymbol{b} \\ -\frac{1}{k}\boldsymbol{b}^T\boldsymbol{A}^{-1} & \frac{1}{k} \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{A}^{-1} + \frac{1}{k}\boldsymbol{A}^{-1}\boldsymbol{b}\boldsymbol{b}^T\boldsymbol{A}^{-1} & -\frac{1}{k}\boldsymbol{A}^{-1}\boldsymbol{b} \\ -\frac{1}{k}\boldsymbol{b}^T\boldsymbol{A}^{-1} & \frac{1}{k} \end{bmatrix}$$

where: $k = c - \boldsymbol{b}^T\boldsymbol{A}^{-1}\boldsymbol{b}$

We can easily simplify this equation if the matrix $\boldsymbol{A}$ is symmetrical as:

$$\boldsymbol{\xi} = \boldsymbol{A}^{-1}\boldsymbol{b} \qquad k = c - \boldsymbol{\xi}^{\boldsymbol{T}}\boldsymbol{b}$$

$$\boldsymbol{M}^{-1} = \frac{1}{k}\begin{bmatrix} k\boldsymbol{A}^{-1} + \boldsymbol{\xi} \otimes \boldsymbol{\xi}^{\boldsymbol{T}} & -\boldsymbol{\xi} \\ -\boldsymbol{\xi}^{\boldsymbol{T}} & 1 \end{bmatrix}$$

where: $\boldsymbol{\xi} \otimes \boldsymbol{\xi}^{\boldsymbol{T}}$ means the tensor multiplication. It can be seen that all computations needed are of $O(N^2)$ computational complexity.

It means that we can compute an inverse matrix incrementally with $O(N^2)$ complexity instead of $O(N^3)$ complexity required originally in this specific case. It can be seen that the structure of the matrix $\boldsymbol{M}$ is "similar to the matrix of the RBF specification.

Now, there is a question how the incremental computation of an inverse matrix can be used for RBF interpolation?

We know that the matrix $\boldsymbol{A}$ in the equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is symmetrical and non-singular if appropriate rules for RBFs are kept.

Point Insertion

Let us imagine a simple situation. We have already computed the interpolation for *N* points and we need to include a new point into the given data set. A brute force approach of full RBF computation on the new data set can be applied with $O(N^3)$ complexity computation.

Let us consider RBF interpolation for *N+1* points and the following system of equations is obtained:

$$\begin{bmatrix} \phi_{1,1} & \phi_{1,N} & \phi_{1,N+1} & x_1 & y_1 & 1 \\ \vdots & \cdot\cdot & \vdots & \vdots & \vdots & 1 \\ \phi_{N,1} & \phi_{N,N} & \phi_{N,N+1} & x_N & y_N & 1 \\ \phi_{N+1,1} & \phi_{N+1,N} & \phi_{N+1,N+1} & x_{N+1} & y_{N+1} & 1 \\ x_1 & x_N & x_{N+1} & 0 & 0 & 0 \\ y_1 & y_N & y_N & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_N \\ \lambda_{N+1} \\ a_x \\ a_y \\ a_0 \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_N \\ f_{N+1} \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \text{where: } \phi_{i,j} = \phi_{j,i}$$

Reordering the equations above we get:

$$\begin{bmatrix} 0 & 0 & 0 & x_1 & x_N & x_{N+1} \\ 0 & 0 & 0 & y_1 & y_N & y_{N+1} \\ 0 & 0 & 0 & 1 & 1 & 1 \\ x_1 & y_1 & 1 & \phi_{1,1} & \phi_{1,N} & \phi_{1,N+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & 1 & \phi_{N,1} & \phi_{N,N} & \phi_{N,N+1} \\ x_{N+1} & y_{N+1} & 1 & \phi_{N+1,1} & \phi_{N+1,N} & \phi_{N+1,N+1} \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_0 \\ \lambda_1 \\ \vdots \\ \lambda_N \\ \lambda_{N+1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ f_1 \\ \vdots \\ f_N \\ f_{N+1} \end{bmatrix}$$

We can see that last row and last column is "inserted". As RBF functions are symmetrical the recently derived formula for iterative computation of the inverse function can be used. So the RBF interpolation is given by the matrix ***M*** as

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^T & c \end{bmatrix}$$

where the matrix ***A*** is the RBF matrix (*N+3*)×(*N+3*) and the vector ***b*** *(N+3)* and scalar value *c* are defined as:

$$\boldsymbol{b} = [x_{N+1} \quad y_{N+1} \quad 1 \quad \phi_{1,N+1} \quad \cdot\cdot \quad \phi_{N,N+1}]^T$$

$$c = \phi_{N+1,N+1}$$

It means that we know how to compute the matrix $\boldsymbol{M}^{-1}$ if the matrix $\boldsymbol{A}^{-1}$ is known.

That is exactly what we wanted!

Recently we have proved that iterative computation of inverse function is of $O(N^2)$ complexity, which offers a significant performance improvement for points insertion. It should be noted that some operations can be implemented more effectively, especially $\boldsymbol{\xi}\otimes\boldsymbol{\xi}^{\boldsymbol{T}} = \boldsymbol{A}^{-1}\boldsymbol{b}\boldsymbol{b}^T\boldsymbol{A}^{-1}$ as the matrix $\boldsymbol{A}^{-1}$ is symmetrical etc.

Point Removal

In some cases it is necessary to remove a point from the given data set. It is actually an inverse operation to the insertion operation described above. Let us consider a matrix ***M*** of the size *(N+1)×(N+1)* as

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^T & c \end{bmatrix}$$

Now, the inverse matrix $\boldsymbol{A}^{-\boldsymbol{1}}$ is known and we want to compute matrix $\boldsymbol{A}^{-1}$, which is of the size *N×N*. Recently we derived opposite rule:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{b}^T & c \end{bmatrix}$$

$$\boldsymbol{\xi} = \boldsymbol{A}^{-1}\boldsymbol{b} \qquad k = c - \boldsymbol{\xi}^{\boldsymbol{T}}\boldsymbol{b}$$

$$\boldsymbol{M}^{-1} = \begin{bmatrix} \boldsymbol{A}^{-1} + \frac{1}{k}\boldsymbol{\xi}\otimes\boldsymbol{\xi}^{\boldsymbol{T}} & -\frac{1}{k}\boldsymbol{\xi} \\ -\frac{1}{k}\boldsymbol{\xi}^{\boldsymbol{T}} & \frac{1}{k} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_{11} & \boldsymbol{Q}_{12} \\ \boldsymbol{Q}_{21} & \boldsymbol{Q}_{22} \end{bmatrix}$$

It can be seen that

$$\boldsymbol{Q}_{11} = \boldsymbol{A}^{-1} + \frac{1}{k}\boldsymbol{\xi}\otimes\boldsymbol{\xi}^{\boldsymbol{T}}$$

and therefore

$$\boldsymbol{A}^{-1} = \boldsymbol{Q}_{11} - \frac{1}{k}\boldsymbol{\xi}\otimes\boldsymbol{\xi}^{\boldsymbol{T}}$$

Now we have both operations, i.e. insertion and removal, with effective computation of $O(N^2)$ computational complexity instead of $O(N^3)$. It should be noted that vectors related to the point assigned for a removal must be in the last row and last column of the matrix $\boldsymbol{M}^{-1}$.

Point selection

As the number of points within the given data set could be high, the point removal might be driven by a requirement of removing a point which causes a ***minimal error of the interpolation***. This is a tricky requirement as there is probably no general answer. The requirement should include additional information which interval of ***x*** is to be considered. Generally we have a function

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \lambda_i \, \phi_i(\boldsymbol{x}) + P_k(\boldsymbol{x})$$

and we want to remove a point $\boldsymbol{x}_j$ which causes a minimal error ε_j of interpolation, i.e.

$$f_j(\boldsymbol{x}) = \sum_{i=1, i \neq j}^{N} \lambda_i \, \phi_i(\boldsymbol{x}) + P_k(\boldsymbol{x})$$

and we want to minimize

$$\varepsilon_j = \int_{\Omega} \left| f(\boldsymbol{x}) - f_j(\boldsymbol{x}) \right| d\boldsymbol{x}$$

where Ω is the interval on which the interpolation is to be made. It means that if the point $\boldsymbol{x}_j$ is removed the error ε_j is determined as:

$$\varepsilon_j = \lambda_j \int_{\Omega} \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) d\boldsymbol{x}$$

As we know the interval Ω on which the interpolation is to be used, we can compute or estimate the error ε_j for each point $\boldsymbol{x}_j$ in the given data set and select the best one. For many functions ϕ the error ε_j can be computed or estimated analytically as the evaluation of ε_j is simple for many functions, e.g.

$$\int r^m \ln dr = r^{m+1} \frac{\ln r}{m+1} - \frac{1}{(m+1)^2}$$

It means that for TPS function $r^2 \ln r$ the error ε_k is easy to evaluate. In the case of CSRBF the estimation is even simpler as they have a limited influence, so generally λ_j determines the error ε_j.

It should be noted, that a selection of a point with the lowest influence to the interpolation precision in the given interval Ω is of $O(N)$ complexity only.

We have shown a novel approach to RBF computation which is convenient for larger data sets. It is especially convenient for t-varying data and for applications, where a "sliding window" is used. Basic operations – point insertion and point removal – have been introduced. These operations have $O(N^2)$ computational complexity only, which makes a significant difference from the original approach used for RBFs computation.

6 RBF Approximation

The RBF interpolation relies on solution of a LSE $\boldsymbol{Ax} = \boldsymbol{b}$ of the size $M \times M$ in principle, where M is a number of the data processed. If the "global" functions are used, the matrix $\boldsymbol{A}$ is full, while if the "local" functions are used (CSRBF), the matrix $\boldsymbol{A}$ is sparse.

However, in visualization applications it is necessary to compute the final function $f(\boldsymbol{x})$ many many times and even for already computed λ_i values, the computation of $f(\boldsymbol{x})$ is too expensive. Therefore it is reasonable to significantly "reduce" the dimensionality of the LSE $\boldsymbol{Ax} = \boldsymbol{b}$. Of course, we are now changing the interpolation property of the RBF to approximation, i.e. the values computed do not pass the given values exactly.

Probably the best way is to formulate the problem using the Least Square Error approximation. Let us consider the formulation of the RBF interpolation again.

$$f(\boldsymbol{x}_i) = \sum_{j=1}^{M} \lambda_j \, \varphi(\|\boldsymbol{x}_i - \boldsymbol{\xi}_j\|) + \boldsymbol{a}^T \boldsymbol{x}_i + a_0$$

$$h_i = f(\boldsymbol{x}_i) \qquad i = 1, \dots, N$$

where: $\boldsymbol{\xi}_j$ are not given points, but points in a pre-defined "virtual mesh" as only coordinates are needed (there is no tessellation needed). This "virtual mesh" can be irregular, orthogonal, regular, adaptive etc. For simplicity, let us consider the two-dimensional squared (orthogonal) mesh in the following example. Then the $\boldsymbol{\xi}_j$ coordinates are the corners of this mesh. It means that the given scattered data will be actually "re-sampled", e.g. to the squared mesh.

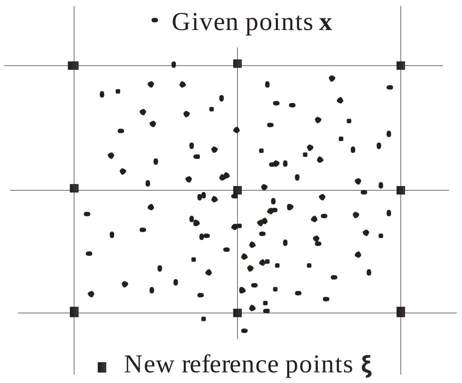

Fig.2. RBF approximation and points' reduction

In many applications the given data sets are heavily over sampled, or for the fast previews, e.g. for the WEB applications, we can afford to "down sample" the given data set. Therefore the question is how to reduce the resulting size of LSE.

Let us consider that for the visualization purposes we want to represent the final potential field in k -dimensional space by P values instead of M and $P \ll M$. The reason is very simple as if we need to compute the function $f(\boldsymbol{x})$ in many points, the formula above needs to be evaluated many times. We can expect that the number of evaluation Q can be easily requested at 10^2 M of points (new points) used for visualization.

If we consider that $Q \geq 10^2\, M$ and $M \geq 10^2\, P$ then **the speed up factor in evaluation can be easily about 10^4** !

This formulation leads to a solution of a linear system of equations $\boldsymbol{Ax} = \boldsymbol{b}$ where number of rows $M \gg P$, number of unknown $[\lambda_1, \dots, \lambda_P]^T$. As the application of RBF is targeted to high dimensional visualization, it should be noted that the polynomial is not requested for all kernels of the RBF interpolation. However it is needed for $\varphi(r) = r^2 lg\, r$ kernel function (TPS).

This reduces the size of the over determined linear system of equations $\boldsymbol{Ax} = \boldsymbol{b}$ significantly. Such system can be solved by the Least Square Method (LSM) as $\boldsymbol{A}^T\boldsymbol{Ax} = \boldsymbol{A}^T\boldsymbol{b}$ or Singular Value Decomposition (SVD) can be used.

$$\begin{bmatrix} \varphi_{1,1} & \cdots & \varphi_{1,P} \\ \vdots & \ddots & \vdots \\ \varphi_{i,1} & \cdot\cdot & \varphi_{i,P} \\ \vdots & \ddots & \vdots \\ \varphi_{M,1} & \cdots & \varphi_{M,P} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_P \end{bmatrix} = \begin{bmatrix} h_1 \\ \vdots \\ \vdots \\ \vdots \\ h_M \end{bmatrix} \qquad \boldsymbol{Ax} = \boldsymbol{b}$$

The high dimensional data can be approximated for visualization by RBF efficiently with a high flexibility as it is possible to add additional points of an area of interest to the mesh. It means that a user can add some points to already given mesh and represent easily some details if requested. It should be noted that the use of LSM increases instability of the LSE in general.

7 Experimental Evaluation

The RBF interpolation is a very powerful tool for interpolation of data in k -dimensional space in general. In order to demonstrate the functionality the RBF, we have recently used RBF for reconstruction of damaged images by a noise or by inpainting [26], [28]. Also a surface reconstruction has been solved by the RBF interpolation well. Fig.3a and Fig.3b illustrates the power of the RBF interpolation [8], [15], [24] for corrupted image reconstruction. The RBF interpolation gives quite good results even if the images are heavily damaged.

The advantages of RBF interpolation over the other interpolations have been proved even though that the RBF interpolation causes some additional computational cost as the RBF is primarily targeted for scattered data interpolation. Fig.4 presents speed of the "standard" and incremental solution and Fig.5 resents the actual speed-up of computation for one inserted or deleted point.

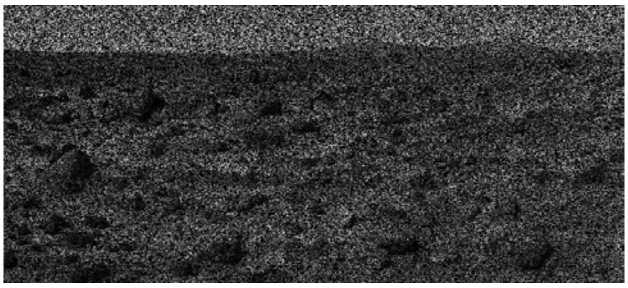
Fig.3a. Original image with 60% of damaged pixels

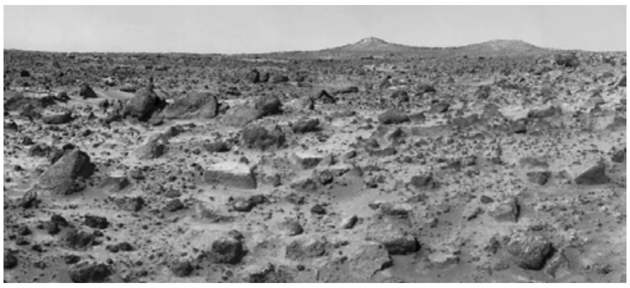
Fig.3b. Reconstructed image [13]

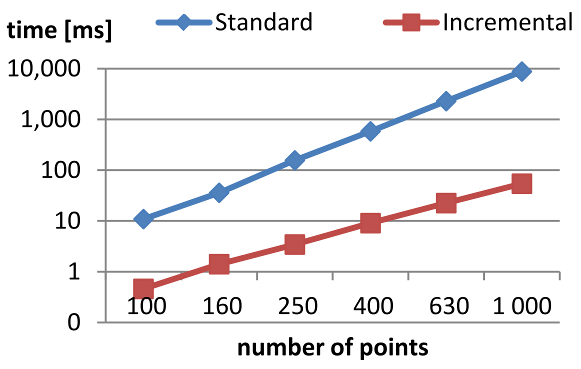

Fig.4: Comparison of "standard" and incremental method

Speed-up is defined as

$$v = \frac{time_{standard}}{time_{incremental}}$$

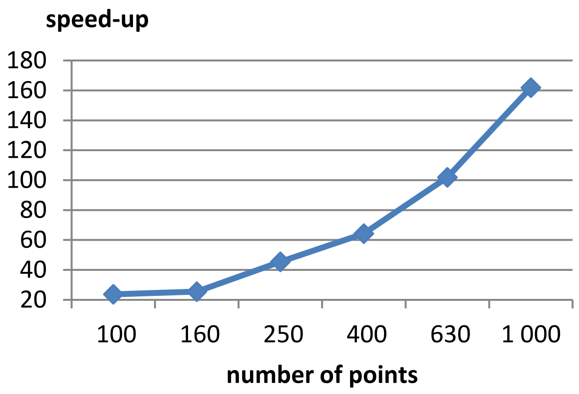

Fig.5: Speed-up of the incremental method

It can be seen that the incremental approach is much faster as expected as the incremental computation is of $O(n^2)$ while the total re-computation of the RBF interpolation is of $O(n^3)$ complexity.

8 Conclusion

The radial basis functions (RBF) interpolation is a representative interpolation method for un-ordered scattered data sets. It is well suited approach for solving problems without meshing the data domain. RBF interpolations are used in many computational fields, e.g. in solution of partial differential equations, DEMs and support the k -dimensional space naturally.

This paper briefly describes a principle of the RBF incremental computation and shows the decrease of the computational complexity from approx. $O(N^3)$ to $O(N^2)$ for a point insertion and a point removal.

The paper also presents a method for "resampling" the data processed as the approximation is acceptable in many applications, namely in visualization. This approach enables to increase details for visualization by adding new points to the "virtual mesh", if more details are needed. It is necessary to mention, that there is no mesh actually needed nor generated and only points of the "virtual mesh" need to be defined.

Acknowledgment: The author thanks to colleagues at the University of West Bohemia (UWB) in Plzen and at the VSB-Technical University (VSB) in Ostrava for their critical comments and constructive suggestions, to anonymous reviewers for their critical view and comments that helped to improve the manuscript. Special thanks belong to former PhD and MSc. students at the UWB Vit Ondracka, Lukas Loukota, Karel Uhlir, Jiri Zapletal and to Jan Hobza for his implementation on GPU and verification of the GPU based image reconstruction.

The project was supported by the MSMT CR, projects No.LH12181 and ME10060.

References:

[1] B. J. Ch. Baxter, The *Interpolation Theory of Radial Basis Functions*, PhD thesis, Trinity College, Cambridge University, U.K., 1992

[2] M. Bertalmio, G. Sapiro, C. Ballester, V. Caselles, Image Inpainting, *Proceedings of SIGGRAPH'00, Computer Graphics*, New Orleans, 23-28, pp.417-424, 2000

[3] J. C. Carr, R. K. Beatson, J. B. Cherrie, T.J. Mitchell, W.R. Fright, B. C. McCallum, T. R. Evans, Reconstruction and representation of 3D objects with radial basis functions, Computer Graphics, *SIGGRAPH 2001 proceedings*, pp. 67-76, 2001.

[4] J. Duchon, Splines minimizing rotation-invariant semi-norms in Sobolev space, *Constructive Theory of Functions of Several Variables*, Springer Lecture Notes in Math, 21, pp. 85-100, 1977

[5] L. R. Hardy, Multiquadric equations of topography and other irregular surfaces, *J. Geophy. Res.*, 76,pp. 1905-1915, 1971

[6] C. A. Micchelli, Interpolation of scattered data: distance matrices and conditionally positive definite functions, *Constr. Approx.*, No.2, pp. 11-22, 1986

[7] S. Jakobsson, B. Andersson, F. Edelvik, Rational radial basis function interpolation with applications to antenna design, *Journal of Computational and Applied Mathematics*, Vol. 233(4), pp. 889-904, December 2009.

[8] R. Pan, V. Skala, Implicit surface modeling suitable for inside/outside tests with radial basis functions, *10th International Conference on Computer Aided Design and Computer Graphics*,DOI.10.1109/CADCG.2007.4407842 pp.28, 2007

[9] R. Pan, V. Skala, A two level approach to implicit modeling with compactly supported radial basis functions, *Engineering and Computers*, ISSN 0177-0667, Vol.27. No.3., pp.299-307, Springer, 2011

[10] R. Pan, V. Skala, Continuous Global Optimization in Surface Reconstruction from an Oriented Point Cloud, *Computer Aided Design*, ISSN 0010-4485, Vol.43, No.8, pp.896-901, Elsevier, 2011

[11] R. Pan, V. Skala, Surface Reconstruction with higher-order smoothness, *The Visual Computer*, ISSN 0178-2789, Vol.28, No.2., pp.155-162, Springer, 2012

[12] I. P. Schagen, Interpolation in Two Dimension – A New Technique, *J. Inst. Maths Applics*, 23, pp. 53-59, 1979

[13] K. Uhlir, V. Skala, Radial basis function use for the restoration of damaged images, *Computer vision and graphics*. Dordrecht: Springer, ISSN 1381-6446, pp. 839-844. 2006

[14] Ch. C. L.Wang, T.-H. Kwok, Interactive Image Inpainting using DCT Based Exemplar Matching, *ISVC 2009*, LNCS 5876, pp.709-718, 2009

[15] W. Wendland, Computational aspects of radial basis function approximation, in K. Jetter et al. (eds.) *Topics in Multivariate Approximation and Interpolation*, Elsevier, pp. 231-256, 2005.

[16] G. B. Wright, *Radial Basis Function Interpolation: Numerical and Analytical Developments*, University of Colorado, PhD Thesis, 2003.
[17] J. Zapletal, P. Vanecek, V. Skala, RBF-based Image Restoration Utilizing Auxiliary Points, *CGI 2009 proceedings*, ACM, ISBN 978-60558-687-8, pp.39-44, 2009.
[18] Y. Ohtake, A. Belyaev, H.-P Seidel, 3D Scattered Data Interpolation and Approximation with Multilevel Compactly Supported RBFs, *Graphical Models*, Vol.67, No.3., pp.150-165, 2005.
[19] V. Skala, Barycentric Coordinates Computation in Homogeneous Coordinates, *Computers & Graphics*, Elsevier, ISSN 0097-8493, Vol. 32, No.1, pp.120-127, 2008
[20] V. Skala, Geometric Computation, Duality and Projective Space, *IW-LGK workshop proceedings*, pp.105-111, Dresden University of Technology, 2011
[21] V.Skala, V. Ondracka, A Precision of Computation in the Projective Space, *Recent Researches in Computer Science*, 15th WSEAS Int.Conference on Computers, Corfu, pp.35-40, ISBN 978-1-61804-019-0, Greece, 2011
[22] V. Skala, Incremental Radial Basis Function Computation for Neural Networks, *WSEAS Transactions on Computers*, Issue 11, Vol.10, pp. 367-378, ISSN 1109-2750,2011.
[23] V. Skala, Robust Computation in Engineering, Geometry and Duality – *TUTORIAL*, *7th Int.Conf. on Systems*, ICONS 2012, St. Gilles, Reunion Island, IARIA, 2011
[24] V. Skala, Interpolation and Intersection Algorithms and GPU, *ICONS 2012*, Saint Gilles, Reunion Island, IARIA, ISBN: 978-1-61208-184-7, pp. 193-198, 2012
[25] V. Skala, Radial Basis Functions for High Dimensional Visualization, *VisGra - ICONS 2012*, Saint Gilles, Reunion Island, IARIA, ISBN: 978-1-61208-184-7, pp. 218-222, 2012
[26] V. Skala, Radial Basis Functions: Interpolation and Applications - An Incremental Approach, *Applied Mathematics, Simulation and Modeling - ASM 2010 conference*, NAUN, ISSN 1792-4332, ISBN 978-960-474-210-3, Greece, pp.209-213, 2010
[27] V. Skala, Duality and Intersection Computation in Projective Space with GPU support, *Applied Mathematics, Simulation and Modeling - ASM conference*, NAUN, pp.66-71, ISSN 1792-4332, ISBN 978-960-474-210-3, Greece,, 2010
[28] V. Skala, Progressive RBF Interpolation, *Afrigraph 2010*, pp.17-20, ACM, ISBN:978-1-4503-0118-3, 2010
[29] D. Coppersmith and S. Winograd, Matrix Multiplication via Arithmetic Programming, *J.Symb. Comput.* 9, 251-280, 1990.
[30] V. Strassen, Gaussian Elimination is Not Optimal, *Numerische Mathematik* 13, 354-356, 1969.

WEB references
[W1] FastRBF: http://www.farfieldtechnology.com/. <retrieved: 2012-08-17>
[W2] Strassen's Algorithm: http://en.wikipedia.org/wiki/Strassen_algorithm <retrieved: 2012-08-17>
[W3] http://mathworld.wolfram.com/StrassenFormulas.html <retrieved: 2012-08-17>

Appendix A

It should be noted that a result of multiplication of two symmetrical matrices is not generally a symmetrical matrix, e.g.

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} d & e \\ e & f \end{bmatrix} = \begin{bmatrix} ad + be & ae + bf \\ bd + ce & be + cf \end{bmatrix}$$

Appendix B

Multiplication $\boldsymbol{C}_{3,2} = \boldsymbol{A}_{3,3} \times \boldsymbol{B}_{3,2}$

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} \\ c_{31} &= a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} \end{aligned}$$

$$\begin{aligned} c_{12} &= a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \\ c_{32} &= a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} \end{aligned}$$

If the matrix $\boldsymbol{A}$ is symmetrical, then only the upper triangular part is to be stored in a linear structure as $a_{ik} = a_{ki}$. This also simplifies the multiplication algorithm for

$$\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$$