Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 433–440

Surface Reconstruction from Scattered Point via RBF Interpolation on GPU

Salvatore Cuomo*, Ardelio Galletti†, Giulio Giunta†, Alfredo Starace†
*Department of Mathematics and Applications "R. Caccioppoli" University of Naples Federico II
c/o Universitario M.S. Angelo 80126 Naples Italy
email:salvatore.cuomo@unina.it
†Department of Applied Science. University of Naples "Parthenope".
Centro Direzionale, Isola C4 80143 Naples Italy
emails:{ardelio.galletti,giulio.giunta}@uniparthenope.it, alfredo.starace@gmail.com

***Abstract*—In this paper we describe a parallel implicit method based on radial basis functions (RBF) for surface reconstruction. Practical applicability of RBF methods is hindered by their high computational demand, that requires the solution of linear systems of size equal to the number of data points. The implementation of our method relies on parallel scientific libraries and is designed for exploiting Graphic Processor Units (GPUs) acceleration. The performance of the proposed method in terms of accuracy of the reconstruction and computing time shows that RBF interpolation can be very effective for large scale surface reconstruction problems.**

I. Introduction

MANY applications in engineering and science need to build accurate digital models of real-world objects defined in terms of *point cloud data*, *i.e.* a set of scattered points in 3D. Typical examples include the digitalization of manufactured parts for quality control, statues and artifacts in archeology and arts [12], human bodies for movies or video games, organs and anatomical parts for medical diagnostic [4] and terrain elevation models for simulations and modeling [16]. Modern 3D scanners are able to acquire point clouds containing millions of points sampled from an object. The process of building a geometric model from such point clouds is usually referred to as *surface reconstruction*.

There are several approaches to reconstruct surfaces from 3D scattered datasets. Generally, such methods fall into two categories [17]: Delaunay-based methods and implicit surface methods. Delaunay triangulation, and other related approaches like Voronoi diagrams, are widely used in digital elevation modeling, and their core numerical problem is a nearest neighbor interpolation [2]. Implicit surface modeling is mostly popular in describing complex shapes and interactive graphical operations. Level set methods [24], moving least square methods [11], variational implicit surfaces [21] and adaptively sampled distance field [9] are recent developments in that field. In this paper, we present an implicit surface method based on radial basis functions (RBFs). In the 1980's, Franke [8] firstly used radial basis functions to interpolate scattered point cloud and proved accuracy and stability of such methods. In this approach, an implicit surface is constructed by calculating the weights of a linear combination of a set of radial basis functions that interpolates the given data points.

Practical applicability of RBF methods is hindered by their computational demand, since they require the solution of a linear system of size equal to the number of data points and current 3D data scanners allow the acquisition of tens of millions points of an object surface.

High Performance Computing is a natural solution to provide the computational power required in such large scale problems [15]. Here, we propose an RBF surface reconstruction method designed for a massively multi-core architecture, namely Graphics Processing Units (GPUs) [18]. Recently, GPUs have been effectively exploited to improve the performance of software tools in several scientific applications, such as computational fluid dynamics, molecular dynamics, climate modeling [6], [7], [10]. The core of our method is the construction of an RBF interpolant. We are not aware of RBF interpolation algorithms for GPU. To our knowledge, the most efficient parallel algorithm for RBF interpolation on multiprocessor clusters is PetRBF [22]. PetRBF exhibits $\mathcal{O}(N)$ complexity, requires $\mathcal{O}(N)$ storage, and scales excellently up to a thousand processes. Our proposed method relies on PetRBF and one of our main contributions consist in developing an improved version of PetRBF for surface reconstruction and GPU acceleration. Moreover, we show how to make a suitable choice of the algorithm parameters for accurate reconstruction from synthetic, real or incomplete datasets. The paper is organized as follows. In Section II we deal with some mathematical preliminaries about implicit surfaces and RBF interpolation. In Section III, first we briefly recall main features of PetRBF, then we describe our method and illustrate our GPU implementation strategy. Section IV contains a discussion on the results of numerical experiments for assessing accuracy and performance of the method. Final conclusions are reported in Section V.

II. Preliminaries

In this section we recall basic ideas underlying implicit surface reconstruction and define the related RBF interpolation problem.

978-1-4673-4471-5/$25.00

A. Implicit Surface Reconstruction

Given a point cloud

$$\mathcal{X} := \{(x_i, y_i, z_i) \in \mathbb{R}^3, i = 1, \ldots, N\}$$

belonging to an unknown surface $\mathcal{M}$, *i.e.* $\mathcal{X} \subset \mathcal{M}$, the goal is to find another surface $\mathcal{M}^*$ which is a reconstruction of $\mathcal{M}$. In the implicit surface approach, $\mathcal{M}$ is defined as the surface of all points $(x, y, z) \in \mathbb{R}^3$ that satisfy the implicit equation

$$f(x, y, z) = 0 \tag{1}$$

for an unknown function f. A way to approximate f is to impose the interpolation conditions (1) on the point cloud $\mathcal{X}$. However, the use of those interpolation conditions only leads to the trivial solution given by the identically zero function, whose zero surface is $\mathbb{R}^3$. Therefore, the key for finding an approximation of the function f is to use additional significant interpolation conditions, *i.e.* involving from off-surface points (where $f \neq 0$). This ensures the existence of a non trivial interpolant $\mathcal{P}_f$, whose zero surface contains a meaningful surface $\mathcal{M}^*$. This approach leads to a surface reconstruction method which consists of three main steps:

1) off-surface points generation;
2) interpolant model identification on the extended dataset;
3) computation of the zero iso-surface of the interpolant.

1) Off-surface points generation:
A common practice [20] is to consider the set of surface normals $\mathbf{n}_i = (n_i^x, n_i^y, n_i^z)$ to the surface $\mathcal{M}$ at points $\mathbf{x}_i = (x_i, y_i, z_i)$. If these normals are not explicitly known, there are techniques and tools[1] that allow to estimate them. Given the oriented surface normals ($\mathbf{n}_i$ and $-\mathbf{n}_i$), the extra off-surface points can be generated by marching a small distance δ along the normals. So for each data point, $\mathbf{x}_i = (x_i, y_i, z_i)$ two additional off-surface points are obtained. The first lies "outside" the surface $\mathcal{M}$ and is given by

$$\begin{aligned}(x_{N+i}, y_{N+i}, z_{N+i}) &= \mathbf{x}_i + \delta \mathbf{n}_i = \\ &= (x_i + \delta n_i^x, y_i + \delta n_i^y, z_i + \delta n_i^z);\end{aligned}$$

the second point lies "inside" and is given by

$$\begin{aligned}(x_{2N+i}, y_{2N+i}, z_{2N+i}) &= \mathbf{x}_i - \delta \mathbf{n}_i = \\ &= (x_i - \delta n_i^x, y_i - \delta n_i^y, z_i - \delta n_i^z).\end{aligned}$$

The union of the sets $\mathcal{X}_\delta^+ = \{\mathbf{x}_{N+1}, \ldots, \mathbf{x}_{2N}\}$, $\mathcal{X}_\delta^- = \{\mathbf{x}_{2N+1}, \ldots, \mathbf{x}_{3N}\}$, and $\mathcal{X}$ gives the overall set of points on which the interpolation conditions are assigned (see Fig. 1). The set $\mathcal{X}_\delta^+$ implicitly defines a surface $\mathcal{M}_\delta^+$ which passes through its points. Analogously, $\mathcal{X}_\delta^-$ defines the surface $\mathcal{M}_\delta^-$. Those two surfaces can be considered as external and internal to $\mathcal{M}$, respectively. The value of δ represents a small step size, whose specific magnitude may be rather critical for a good surface reconstruction [3]. In particular, if δ is chosen too large, this results in self intersecting $\mathcal{M}_\delta^+$ or $\mathcal{M}_\delta^-$ auxiliary surfaces. In our implementation we fix δ to 1% of the side of the bounding box of the data, as suggested in [23].

[1]package ply.tar.gz provided by Greg Turk, available at http://www.cc.gatech.edu/projects/large_models/ply.html

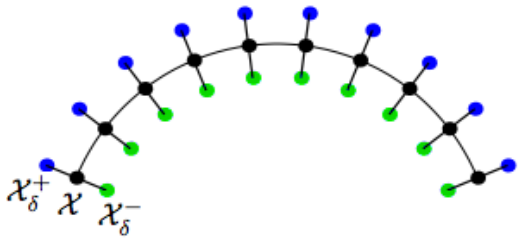

Fig. 1. Extended data set. In black points from $\mathcal{X}$, in blue points from $\mathcal{X}_\delta^+$ and in green points from $\mathcal{X}_\delta^-$

2) Interpolant model identification on the extended dataset:

This step consists in determining a function $\mathcal{P}_f$ whose zero contour (iso-surface $\mathcal{P}_f = 0$) interpolates the given point cloud data $\mathcal{X}$, and whose iso-surface $\mathcal{P}_f = 1$ and $\mathcal{P}_f = -1$ interpolate $\mathcal{X}_\delta^+$ and $\mathcal{X}_\delta^-$, respectively, *i.e.*

$$\mathcal{P}_f(x_i) = \begin{cases} 0 & i = 1, \ldots, N \\ 1 & i = N+1, \ldots, 2N \\ -1 & i = 2N+1, \ldots, 3N \end{cases} \tag{2}$$

The values of ± 1 for the auxiliary data are assigned in an arbitrary way. Such choice does not affect the quality of the results. Here we are interested to the zero iso-surface of $\mathcal{P}_f$.

3) Computation of the zero iso-surface of the interpolant:
In order to evaluate the $\mathcal{P}_f$ zero iso-surface and visualize it, we simply evaluate the interpolant $\mathcal{P}_f$ on a dense grid in a bounding box containing the point cloud. This approach leads to some undesired artifacts, since there are points in the bounding box which do not belong to $\mathcal{M}^*$. A way to overcome this drawback and display only $\mathcal{M}^*$ consists in evaluating the interpolant in a small *surrounding* volume of the surface $\mathcal{M}$. This set is denoted as $\mathcal{M}_{ext}^\varepsilon = \{\mathbf{x} \in \mathbb{R}^3 : \ d(\mathbf{x}, \mathcal{M}) \leq \varepsilon\}$, where $d(\mathbf{x}, \mathcal{M}) = \inf_{\mathbf{y} \in \mathcal{M}} \|\mathbf{y} - \mathbf{x}\|$. For a small enough value of ε, it holds that

$$\mathcal{M}^* \approx \mathcal{M}_{ext}^\varepsilon \cap \mathcal{S}_0,$$

where $\mathcal{S}_0$ is the zero iso-surface of $\mathcal{P}_f$.

B. RBF interpolation

Given a set of N distinct points $(x_j,\ y_j)$, $j = 1, \ldots, N$, where $x_j \in \mathbb{R}^s$ and $y_j \in \mathbb{R}$, the scattered data interpolation problem consists in finding an interpolant function $\mathcal{P}_f$ such that:

$$\mathcal{P}_f(x_j) = y_j,\ j = 1, \ldots, N. \tag{3}$$

In the univariate setting ($s = 1$), the interpolant $\mathcal{P}_f$ is usually chosen in a suitable function space. A common approach assumes the function $\mathcal{P}_f$ as a linear combination of certain basis functions B_j

$$\mathcal{P}_f(x) = \sum_{j=1}^{N} c_j B_j(x). \tag{4}$$

In a multivariate setting ($x_j \in \mathbb{R}^s,\ s > 1$), the problem is more complex. As stated by the *Mairhuber-Curtis* theorem [5], [13], in order to have a well-posed multivariate scattered data interpolation problem, it is not possible to fix in advance the basis $\{B_1, \ldots, B_N\}$, since the basis functions must depend on the data sites x_j.

The data dependent space for RBF interpolation can be easily generated by means of the radial functions:

$$B_j \equiv \Phi_j = \varphi(\|x - x_j\|).$$

The points x_j to which the basic function φ is shifted are usually referred to as *centers*. While there may be circumstances that suggest to choose these centers different from the data sites one generally picks the centers to coincide with the data sites.

In fact, a practical interpolation problem consists of two subproblems: finding the interpolant $\mathcal{P}_f$ and evaluating it on an assigned set of points. The coefficients c_j in (4) are obtained by imposing the interpolation conditions (3)

$$\mathcal{P}_f(x_i) = \sum_{j=1}^{N} c_j \varphi(\|x_i - x_j\|) = y_i,\ i = 1, \ldots, N.$$

This leads to solve the linear system of equations $Ax = b$ in (5).

Given a set of M points $\xi = \{\xi_1, \xi_2, \ldots, \xi_M\}$, the evaluation of the interpolant $\mathcal{P}_f$ on ξ can be computed as a matrix-vector product (6).

It is well known that in order to have a well-posed problem (5), the matrix A must be non-singular. Unfortunately, a complete characterization of the class of all basic functions φ that generate a non-singular matrix for an arbitrary set $\mathcal{X} = \{x_1, \ldots, x_N\}$ of distinct data sites is still lacking. The situation gets better in case of *positive definite matrices*, that are always non-singular. Popular radial basis functions Φ_j, that give rise to positive definite interpolation matrices, are summarized in Table II-B. We focused our work on the gaussian function, as in [22].

III. GPU Parallel Surface Reconstruction

A brief description of the surface reconstruction algorithm is reported below:

Algorithm 1 Surface Reconstruction

Requirements:
point cloud $\mathcal{X}$, surface normals $\mathbf{n}_i$,
evaluation grid ξ

1: compute extended data set:
$\mathcal{X}_{ext} = \mathcal{X} \bigcup \mathcal{X}_\delta^+ \bigcup \mathcal{X}_\delta^-$ by using $\mathbf{n}_i$;
2: find the interpolant $\mathcal{P}_f$ on $\mathcal{X}_{ext}$;
3: evaluate $\mathcal{P}_f$ on ξ;
4: render the surface;

Steps 1 and 2 have been already discussed in §II.A. Step 3 requires a matrix-vector multiplication as stated in §II.B and the rendering step can be simply accomplished using either the MATLAB `isosurface` feature or other specific tools. The most computationally expensive step is the second one, that requires the solution of a system of $3N$ linear equation, where N is the initial point cloud size. In the following, we describe a parallel scheme for solving this RBF interpolation linear system.

A. Parallelization scheme

In handling problems with a large number of data points, as in surface reconstruction from clouds of millions of points, the large amount of memory storage can become a critical point. As the problem size grows, parallelization on distributed memory architectures becomes necessary. Domain Decomposition is a useful parallelization strategy for large scale interpolation, since the solution of the original system is built up by solving a set of smaller subproblems that interact through their interfaces. Below, we briefly overview the parallelization strategy of PetRBF, in a 3D setting. Let Ω be a 3D domain containing the point cloud and let partition Ω in overlapping sub-domains $\Omega_k,\ k = 1, \ldots, S$. Moreover, let $\tilde{\Omega}_k$ denote the empty intersection portions of subdomains Ω_k (see Fig. 2). The solution of the RBF interpolation linear system

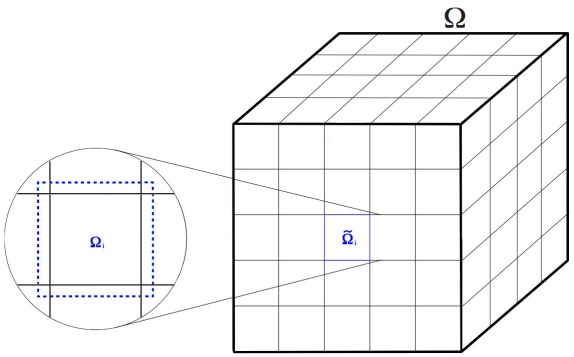

Fig. 2. Illustration of the Domain Decompostion Method.

$Ax = b$ on the whole domain Ω can be obtained through an iterative method consisting in sequentially solving, for any Ω_k, the linear (sub)system $A_k x_{\Omega_k} = b_{\Omega_k}$, where A_k, x_{Ω_k} and b_{Ω_k} are the entries in A, x, and b belonging to the sub-domain Ω_k, respectively. If at each iteration the solution on the entire domain is updated simultaneously after the solutions on every sub-domain are computed, we have an *additive* Schwarz method. Moreover, if the entries of x_{Ω_k} which are outside of the subdomain $\tilde{\Omega}_k$ are discarded after the calculation on each subdomain Ω_k, a *restricted additive* Schwarz method (RASM) in defined. RASM is known to converge faster than additive Schwarz, and requires less communication in a parallel setting. Notice that solving smaller systems of equations has the same effect of a preconditioning technique, and then RASM can be used in combination with any iterative method. PetRBF uses a Krylov subspace methods, namely the Generalized Minimum Residual (GMRES).

If the basis functions exhibit negligible global effects then A can be considered a band matrix, so that the matrix-vector

$$\underbrace{\begin{bmatrix} \varphi(\|x_1-x_1\|) & \varphi(\|x_1-x_2\|) & \cdots & \varphi(\|x_1-x_N\|) \\ \varphi(\|x_2-x_1\|) & \varphi(\|x_2-x_2\|) & \cdots & \varphi(\|x_2-x_N\|) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(\|x_N-x_1\|) & \varphi(\|x_N-x_2\|) & \cdots & \varphi(\|x_N-x_N\|) \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{b} \tag{5}$$

$$\begin{bmatrix} \mathcal{P}_f(\xi_1) \\ \mathcal{P}_f(\xi_2) \\ \vdots \\ \mathcal{P}_f(\xi_M) \end{bmatrix} = \begin{bmatrix} \varphi(\|\xi_1-x_1\|) & \varphi(\|\xi_1-x_2\|) & \cdots & \varphi(\|\xi_1-x_N\|) \\ \varphi(\|\xi_2-x_1\|) & \varphi(\|\xi_2-x_2\|) & \cdots & \varphi(\|\xi_2-x_N\|) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(\|\xi_M-x_1\|) & \varphi(\|\xi_M-x_2\|) & \cdots & \varphi(\|\xi_M-x_N\|) \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} \tag{6}$$

TABLE I
EXAMPLES OF RADIAL BASIS FUNCTIONS.

RBF	Φ	
Poisson radial function	$\frac{J_{s/2-1}(\|x-x_j\|)}{\|x-x_j\|^{s/2-1}}$	$s \geq 2$
Inverse Multiquadric	$(1+\|x-x_j\|^2)^{-\beta}$	$\beta > \frac{s}{2}$
Matèrn function	$\frac{K_\alpha(\|x-x_j\|)\|x-x_j\|^\alpha}{2^{\beta-1}\Gamma(\beta)}$	$\alpha = \beta - \frac{s}{2} > 0$
Whittaker function	$\int_0^{+\infty}(1-\|x-x_j\|)_+^{k-1}t^\alpha e^{-\beta t}dt$	$k = 2,3,\ldots, \quad \alpha = 0,1,\ldots$
Gaussian function	$\frac{1}{\sqrt{2\pi}\sigma}e^{\frac{-\|x-x_j\|^2}{2\sigma^2}}$	$\sigma > 0$

product, which is the predominant operation in GMRES, can be computed somewhat locally. Using a gaussian function as the basic function, it holds that A has the following entries:

$$A_{ij} = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{-\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}\right). \tag{7}$$

Since the gaussian function decays rapidly, for a suitable choice of σ, the entries of A in (6) which correspond to the interaction of distant points can be neglected. Such resulting sparsity of A depends on the relative size of the domain compared to the standard deviation σ of the gaussian. If σ is kept constant while the size of the domain increases with N, the calculation load will scale as $\mathcal{O}(N)$. The communication required to compute $A_k x_{\Omega_k}$ is also restricted to a constant number of entries corresponding to neighbor points. Therefore, RASM entails a high level of parallelism at each GMRS iteration, and a good scalability for surface reconstruction problems with a domain size as large as hundreds (or even thousands) of σ [22].

A remarkable implementation feature of PetRBF is the use of the parallel numerical library PETSc (Portable, Extensible Toolkit for Scientific Computation) [1]. All vectors and matrices can be distributed via PETSc routines in a such way that each process stores only a local portion of data. In particular, this can be done by defining x and b as `Vec` PETSc objects. Overlapping and non-overlapping sub-domains can be handled by means of index sets (`IS`). A PETSc `IS` object is a global index that identifies the entries in each sub-domain and is distributed among the processes in the same way as the vectors. Recalling that the interpolation matrix has entries which depends only on the vector x of data sites and σ in the gaussian (eq. 7), it turns out that A can be represented as a `MatShell` PETSc object, that allows to perform matrix computations in a matrix-free way, i.e. without actually storing the matrix. Inner products, norms, and scalar products are computed by specific PETSc routines, and the solution of the linear system is obtained by calling the `KSPSolve` PETSc solver.

B. GPU implementation

Our basic idea consists in using the new PETSc GPU feature in the latest version of PETSc [1]. In fact, we have developed an improved version of PetRBF that extends the original code to GPU environments. GPU support to PETSc has been introduced by means of the CUDA framework and exploits the open source libraries THRUST [19] and CUSP [14]. THRUST is a collection of data parallel primitives that provide high level abstractions to describe efficient computations on GPU. CUSP is a sparse linear algebra toolkit for performing matrix operations and solving linear systems via CUDA on GPUs. They allows a transparent access to the GPU, without radically changes to the existing source code of PETSc. In PETSc GPU, two new subclasses `VecCUSP` and `MatCUSP` of the `Vector` and `Matrix` classes have been defined, which in turns rely on CUBLAS, CUSP, and THRUST routines to perform matrix and vector operations on the GPU. CUSP natively supports several sparse matrix formats:

- Coordinate list (COO)
- Compressed Sparse Row (CSR)
- Diagonal (DIA)
- ELLPACK (ELL)
- Hybrid (HYB)

It is worth noting that in PETSc GPU by using the routines `VecSetType()` and `MatSetType()` we can select the desired sparse matrix format simply with the command line option `-mat_cusp_storage_format <format>`, and switch from a CPU version to a GPU version by means of the command line parameters `-vec_type cusp` and `-mat_type aijcusp`.

Our GPU parallel implementation of the implicit surface reconstruction method in Algorithm 1 focuses on steps 2 and 3, i.e. construction of the RBF interpolant for the extended dataset (2), and evaluation of such RBF interpolant on a given set of evaluation points. The most expensive operation in step 2 is the matrix-vector multiplication at each iteration of the RASM preconditioned GMRES. The choice of a suitable PETSc object for the sparse matrix A turns out to be crucial for the efficiency of the implementation of step 2 on the GPU. According to the results of specific experiments, we decided to use the Diagonal (DIA) sparse matrix format in CUSP, and the related CUSP sparse matrix operations. Besides taking advantage of very reliable and efficient CUSP routines, the resulting code exhibits a very low overhead for data transfer from host memory (CPU) to device memory (GPU). By contrast, in implementing step 3 we represented the matrix A as a `MatShell` PETSc object and then we used the CUDA kernel reported in Algorithm 3 below. This choice is justified by the fact that the matrix-vector multiplication in the evaluation step must be performed only once, so that the time needed to compute the non-zero entries of A, using the Algorithm 2 as in step 2, would nullify all the advantages of the efficient CUSP routines. Moreover, our CUDA kernel makes use of the shared memory portion of the device memory, whose low latency improves the performance of the computation.

Algorithm 2 Pseudo-code for the construction of the interpolation matrix

1: **for each** subdomain Ω_i **do**
2: **for each** point x_i in the subdomain Ω_i **do**
3: **for each** point x_j in the truncation area of Ω_i **do**
4: Set $A_{ij} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right)$
5: **end for**
6: **end for**
7: **end for**

IV. Experimental results

In this section we present some results of our method for surface reconstruction. The results were computed using a system equipped with an Intel Core i7-940 CPU (2,93 GHz, 8M Cache). The middleware framework is OS Linux kernel 2.6.32-28 and PETSc developer version 3.3.

First, we investigate the impact of the parameter σ on the quality of reconstruction. As we showed in §III.B, our method is very efficient for small values of σ compared to the domain size. However, besides efficiency, even the quality of the result also depends on the values of σ. This is because the accuracy

Algorithm 3 CUDA code for the evaluation step

```
__shared__ float sharedXi[BLOCK_SIZE];
__shared__ float sharedGi[BLOCK_SIZE];
int bx = blockIdx.x;
int tx = threadIdx.x;
int i = blockIdx.x * BLOCK_SIZE +
threadIdx.x;
float pf = 0;
float coeff = 0.5f/(sigma*sigma);
for (unsigned int m = 0; m <
(col-1)/BLOCK_SIZE+1; m++) {
sharedXi[tx] = Xi[m*BLOCK_SIZE + tx];
__syncthreads();
for (unsigned int k = 0; k <
BLOCK_SIZE; k++) {
dx = Xj[i]-sharedXi[k];
pf += sharedGi[k]*exp(-(dx*dx)*coeff;}}
Pf[i] = pf/M_PI*coef;
```

of the interpolation model depends on the ratio between the density of the point cloud and σ.

The experiments carried out in [22] were designed only for equally-spaced lattice point distributions, and the point density was measured by the spacing h between the points. In that case, a good choice of σ, in terms of performance and accuracy, is that yielding $h/\sigma \approx 1$.

For non-uniform unorganized data, more appropriate density measures can be devised. One is the so-called *separation distance* defined as

$$q_{\mathcal{X}} = \frac{1}{2} \min_{i \neq j} \|x_i - x_j\|_2. \tag{8}$$

As shown in Fig. 3, $q_{\mathcal{X}}$ geometrically represents the radius of the largest (hyper)sphere that can be drawn around each point in such a way that no (hyper)sphere intersects the others; that is why it is sometimes called *packing radius*. Another measure, popular in approximation theory, is the so-called *fill distance*:

$$h_{\mathcal{X},\Omega} = \sup_{x \in \Omega} \min_{x_j \in \mathcal{X}} \|x - x_j\|_2. \tag{9}$$

It indicates how well the set $\mathcal{X}$ fills the domain Ω. A geometric interpretation of the fill distance is given by the radius of the biggest empty (hyper)sphere that can be placed among the data sites in Ω (see Fig. 3); for this reason, it is also referred to as *covering radius*. Hence for scattered data, using (8) and (9), the heuristic "optimal" ratio $h/\sigma \approx 1$ can be expressed as follows:

$$\sigma \approx 2q_{\mathcal{X}} \tag{10}$$

and

$$\sigma \approx h_{\mathcal{X},\Omega}\sqrt{2}. \tag{11}$$

A. Tests on synthetic dataset

The experiments on synthetic dataset deal with a sphere, that allows to easily calculate (8) and (9). In order to perform

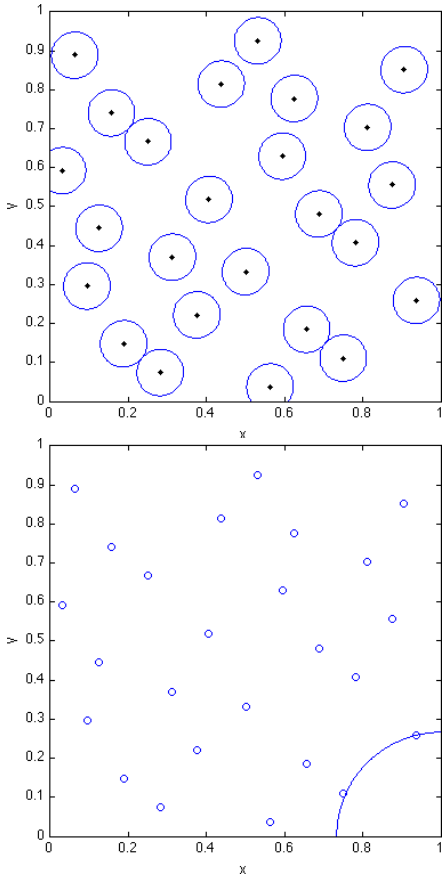

Fig. 3. Geometric interpretation of separation distance (on the left) and fill distance (on the right) for 25 Halton points on the domain $\Omega = [0,1]^2$ ($q_{\mathcal{X}} \approx 0.0597$ and $h_{\mathcal{X},\Omega} \approx 0.2667$).

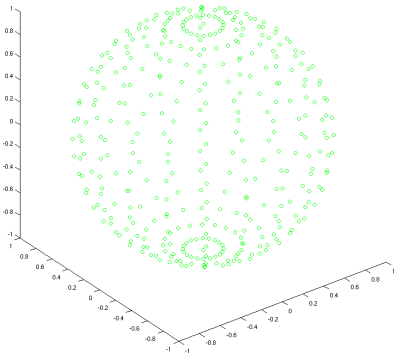

Fig. 4. 382 point cloud from the unit sphere centered in the origin.

a test consistent with a real dataset, the widely scattered point cloud in Fig. 4 was selected.

As shown in Fig. 5 (top left), a too small value of σ leads to a surface that actually interpolates the given point cloud but whose reconstruction quality is unsatisfactory. Notice that, even using (10) as in [22], one couldn't achieve a better result (see Fig. 5, top right). On the contrary, by using (11) and increasing the value of σ (see Fig. 5, bottom left) the quality of the reconstruction improves up to the desired level (see Fig.

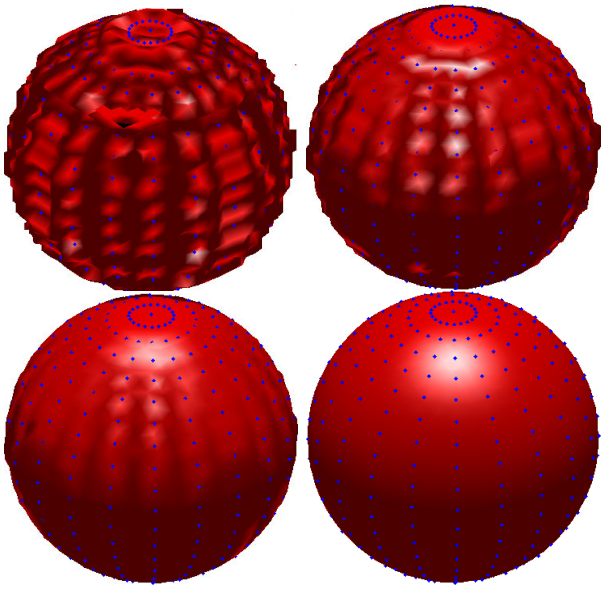

Fig. 5. Reconstructed sphere for different values of σ. (from left to right $\sigma = 0.025$, $\sigma = 2q_{\mathcal{X}} = 0.048$, $\sigma = 0.065$, $\sigma = h_{\mathcal{X},\mathcal{M}}\sqrt{2} = 0.157$)

5, bottom right). It is interesting to note that for intermediate values of σ, though the quality is not globally satisfactory, it appears to be locally adequate, mainly in those parts of the surface where the points are at a distance $d \approx \sigma$.

1) Tests on incomplete data:

We assessed the sensitivity of our method to the lack of information by means of an incomplete dataset. We began with a dataset composed of 50% randomly chosen points from the previous dataset; Fig. 6 shows the new the point cloud. In this case, it turns out that an "optimal" value of σ is

$$\sigma = h_{\mathcal{X},\mathcal{M}}\sqrt{2} = 0.328.$$

This optimal value corresponds to the actual fill distance for the new dataset and provides a successful reconstruction of the sphere (see Fig. 6, bottom). We remark that the old value $\sigma = 0.157$ would lead to a reconstructed surface which is not the desired sphere (see Fig. 6, top).

Another interesting test deals with a point cloud belonging to the (upper) semi-sphere:

$$\mathcal{M}^+ = \{(x,y,z) \in \mathbb{R}^3 : \ x^2 + y^2 + z^2 = 1, z \geq 0\}.$$

If we set $\sigma = h_{\mathcal{X},\mathcal{M}^+}\sqrt{2} = 0.157$, then we can reconstruct the $\mathcal{M}^+$ surface (see Fig.7, top) from which the dataset was extracted. If we assume that the point cloud came from the whole sphere $\mathcal{M}$, then the value of the fill distance would change to $h_{\mathcal{X},\mathcal{M}} = \sqrt{2}$. The latter value of the fill distance gives an optimal value of $\sigma = h_{\mathcal{X},\mathcal{M}}\sqrt{2} = 2$, which leads to the reconstruction of the whole sphere, as shown in Fig. 7, bottom.

B. Tests on real dataset

We briefly discuss the results of some tests concerning real dataset, namely the Stanford Bunny model. This is composed

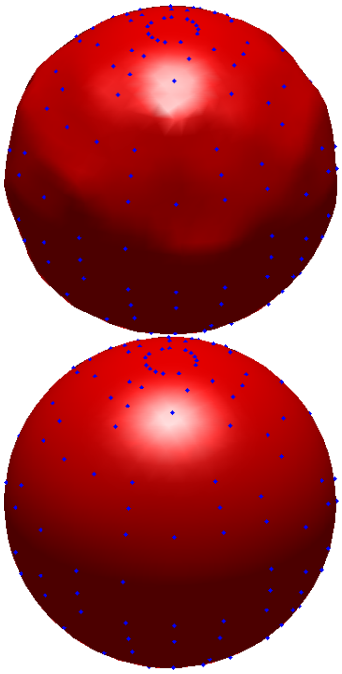

Fig. 6. Reconstructed sphere from incomplete data. (on the left $\sigma = 0.157$ and on the right $\sigma = h_{\mathcal{X},\mathcal{M}}\sqrt{2} = 0.328$)

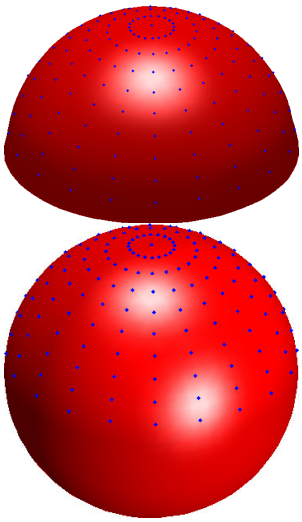

Fig. 7. Reconstructed surface from upper semi-sphere data. (on the left $\sigma = h_{\mathcal{X},\mathcal{M}^+}\sqrt{2} = 0.157$ and on the right $\sigma = h_{\mathcal{X},\mathcal{M}}\sqrt{2} = 2$)

of $N = 8171$ points, giving an extended dataset of $N_{ext} = 3N = 24513$ points. In order to select a suitable value of σ, we need to calculate the value of the fill distance for the given dataset. On real datasets, where the geometry of the surface is either unknown or very complex, this task can be a real challenge. Recalling that the fill distance measures the data density in the membership domain, we introduce a new measure defined as

$$h_{max} = \max_j \min_{i \neq j} \|x_i - x_j\|_2.$$

This represents the largest value of the distances between each point and its nearest neighbor point. For a dataset without multiple *connected components*, the fill distance can be approximated by

$$h_{\mathcal{X},\mathcal{M}} \approx h_{max}\sqrt{2}. \tag{12}$$

As in the case of synthetic dataset, a too small value of σ results in a low quality reconstructed surface (see Fig. 8, top left)), intermediate values lead to reconstructions which are adequate only where there is a high density of points (see Fig.8, top right), while the choice (12) of $\sigma \approx h_{\mathcal{X},\mathcal{M}}$ provides the best result (see Fig. 8, bottom).

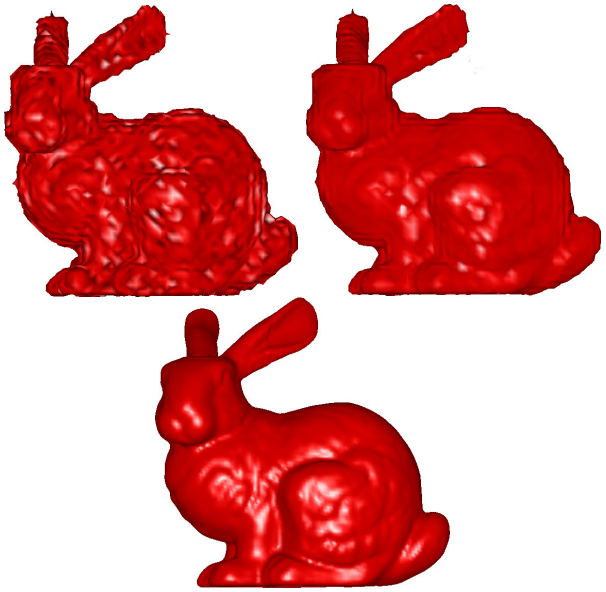

Fig. 8. Reconstructed bunny for different values of σ. (from left to right $\sigma = 0.0007$, $\sigma = 0.0012$ and $\sigma = h_{max} \approx h_{\mathcal{X},\mathcal{M}}\sqrt{2} = 0.0033$)

C. Tests on performance

We report some results on the performance of the GPU implementation of our method compared to its CPU implementation. CPU times refer to the execution on one core of the i7-940 CPU, and GPU times refer to execution on the Nvidia Fermi C1060 GPU, with 4Gb of RAM. The middleware software consists of PETSc developer version 3.3, compiled with GPU support, CUDA release 4.2, CUSP version 0.3.0 and THRUST version 1.5.2.

We have used synthetic point clouds of increasing size number N, i.e with an increasing the density. Since we want to emphasize the advantages in exploiting the GPU, we choose a constant value of σ ($\sigma = 0.157$). Notice that if the value of σ scaled with the density, then the problem would scale as $\mathcal{O}(N)$, giving rise to a less noticeable GPU acceleration.

In Tab. II, we report the execution times on a single CPU and a single GPU, and also the resulting speed-ups for the interpolant construction (step 2 of the reconstruction algorithm), varying the size N of the dataset. To make a fair comparison, we fixed the number of iterations in GMRES to 50. Results

in Tab. II show that, even though the RASM preconditioner is not available in CUSP, the GPU implementation achieves a 6.6 sped-up.

TABLE II
SPEED-UPS FOR THE INTERPOLANT CONSTRUCTION STEP ON GPU.

N	CPU	GPU	Speed up
1323	0,67551	0,1404	4,81
4686	34,567	5,6472	6,12
15625	451,75	71,57	6,31
24036	5398,4	815,09	6,63

In Tab. III we reported the execution times on CPU and GPU, and the resulting speed-ups for the interpolant evaluation (step 3), varying the size N of the point cloud and the size M of the evaluation grid. As expected, these execution times are substantially lower than those obtained for the step 2, but in this case the GPU is fully exploited and larger speed-ups are achieved.

TABLE III
SPEED-UPS FOR THE EVALUATION STEP ON GPU.

$M\backslash N$		1323	4686	15625	24036
15625	CPU	0,11571	0,20605	0,78172	0,97539
	GPU	0,012831	0,033589	0,12167	0,13699
	Speed up	**9,01**	**6,13**	**6,42**	**7,12**
125000	CPU	0,60406	1,5657	5,9558	7,433
	GPU	0,029907	0,09974	0,32034	0,43697
	Speed up	**20,19**	**15,69**	**18,59**	**17,01**
421875	CPU	1,9098	5,2612	19,946	25,037
	GPU	0,084603	0,2892467	0,89695	1,1272
	Speed up	**22,57**	**18,18**	**22,23**	**22,211**
1000000	CPU	4,4389	12,505	47,594	59,501
	GPU	0,18456	0,60741	2,0629	2,4968
	Speed up	**24,05**	**20,58**	**23,07**	**23,83**
1953125	CPU	8,8515	24,431	92,589	116,39
	GPU	0,35668	1,0662	3,9195	4,8035
	Speed up	**24,81**	**22,91**	**23,62**	**24,23**
3375000	CPU	15,018	42,166	160,17	199,45
	GPU	0,59846	1,7525	6,4671	7,9468
	Speed up	**25,09**	**24,06**	**24,76**	**25,09**

V. CONCLUSION

We proposed a parallel method based on radial basis functions for surface reconstruction on GPU. Our implementation relies on GPU-parallel scientific libraries, in order to take full advantage of the computing power of the GPU device. We showed that reconstruction quality and performance of the method are strongly related to the gaussian RBF parameter σ. We proposed an optimal heuristic estimate of such parameter based on suitable density measures of the point cloud. Finally, the observed speed-ups and running times confirm that the RBF interpolation can be a very effective approach to solve large scale surface reconstruction problems.

REFERENCES

[1] Satish Balay, William Gropp, Lois Curfman McInnes, and Barry F Smith. Petsc, the Portable, Extensible Toolkit for scientific computation. *Argonne National Laboratory*, 2:17, 1998.

[2] Alex Beutel, Thomas Mlhave, Pankaj K. Agarwal, Natural neighbor interpolation based grid DEM construction using a GPU. Proc. of the 18th SIGSPATIAL International Conf. on Advances in Geographic Information Systems. pp 172-181. ACM New York, 2010

[3] Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proc. of the 28th annual Conf on Computer graphics and interactive techniques*, pages 67–76. ACM, 2001.

[4] Jonathan C. Carr, W. Richard Fright, and Richard K Beatson. Surface interpolation with radial basis functions for medical imaging. *Medical Imaging, IEEE Transactions on*, 16(1):96–107, 1997.

[5] P.C. Curtis. n-parameter families and best approximation. *Pacific Journal of Mathematics*, 9(4):1013–1027, 1959.

[6] S. Cuomo, P. De Michele, R. Farina, F. Piccialli, A Smart GPU Implementation of an Elliptic Kernel for an Ocean Global Circulation Model, *Applied Mathematical Sciences*, Vol. 7, no. 61, 3007 - 3021 (2013).

[7] S. Cuomo, P. De Michele, R. Farina, A CUBLAS-CUDA implementation of PCG method of an ocean circulation model, *AIP Conf. Proc.*,1389, pp. 1923-1926; doi:http://dx.doi.org/10.1063/1.3636988 (2011).

[8] R. Franke. Scattered data interpolation: Tests of some methods. *Math. Comput.*, 38(157):181–200, 1982.

[9] Sarah F Frisken, Ronald N Perry, Alyn P Rockwood, and Thouis R Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proc. of the 27th annual Conf on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co., 2000.

[10] F. Farina, S. Cuomo, P. De Michele, F.Piccialli, An inverse preconditioner for a free surface ocean circulation model *AIP Conf. Proc.*, 1493, pp. 356-362; doi:http://dx.doi.org/10.1063/1.4765513 (2012).

[11] Jan Klein and Gabriel Zachmann. Point cloud surfaces using geometric proximity graphs. *Computers & Graphics*, 28(6):839–850, 2004.

[12] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proc. of the 27th annual Conf. on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 131–144. ACM Press/Addison-Wesley Publishing Co., 2000.

[13] J.C. Mairhuber. On haar's theorem concerning chebychev approximation problems having unique solutions. *Proc. of the American Mathematical Society*, pages 609–615, 1956.

[14] Nathan Bell and Michael Garland. Cusp: Generic parallel algorithms for sparse matrix and graph computations, 2012. Version 0.3.0.

[15] F. Piccialli, S. Cuomo, P. De Michele, A Regularized MRI Image Reconstruction based on Hessian Penalty Term on CPU/GPU Systems, *Procedia Computer Science*, 18, 2643-2646 (2013).

[16] Joachim Pouderoux, Jean-Christophe Gonzato, Ireneusz Tobor, and Pascal Guitton. Adaptive hierarchical rbf interpolation for creating smooth digital elevation models. In *Proc. of the 12th annual ACM international workshop on Geographic information systems*, GIS '04, pages 232–240. ACM, 2004.

[17] Oliver Schall and Marie Samozino. Surface from scattered points. In *a Brief Survey of Recent Developments. 1st International Workshop on Semantic Virtual Environments, Page (s)*, pages 138–147, 2005.

[18] J Süßmuth, Q Meyer, and G Greiner. Surface reconstruction based on hierarchical floating radial basis functions. In *Computer Graphics Forum*, volume 29, pages 1854–1864. Wiley Online Library, 2010.

[19] Jared Hoberock and Nathan Bell. Thrust: A parallel template library, 2010. Version 1.5.2.

[20] Greg Turk, Huong Quynh Dinh, James F O'Brien, and Gary Yngve. Implicit surfaces that interpolate. In *Shape Modeling and Applications, SMI 2001 International Conf on.*, pages 62–71. IEEE, 2001.

[21] Greg Turk and James F O'brien. Variational implicit surfaces. 1999.

[22] Rio Yokota, L.A. Barba, and Matthew G. Knepley. Petrbf a parallel o(n) algorithm for radial basis function interpolation with gaussians. *Computer Methods in Applied Mechanics and Engineering*, 199(25):1793 – 1804, 2010.

[23] H. Wendland. *Scattered data approximation*, volume 2. Cambridge University Press Cambridge, 2005.

[24] Hong-Kai Zhao, Stanley Osher, and Ronald Fedkiw. Fast surface reconstruction using the level set method. In *Variational and Level Set Methods in Computer Vision, 2001. Proc. IEEE Workshop on*, pages 194–201. IEEE, 2001.