Kernel-Based Meshless Methods

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Andrew Corrigan
Bachelor of Science
Stevens Institute of Technology, 2005

Co-Director: Dr. John Wallin, Professor
Department of Computational and Data Sciences
Co-Director: Dr. Thomas Wanner, Professor
Department of Mathematical Sciences

Spring Semester 2009
George Mason University
Fairfax, VA

# Dedication

I dedicate this dissertation to my family and friends for their love and support.

# Acknowledgments

I would like to thank the following people. My advisors Dr. John Wallin and Thomas Wanner, my committee members Dr. Daniel Anderson, Dr. Juan Cebral, and Dr. David Singman, as well as Dr. Rainald Löhner, for their wisdom, support, encouragement, and guidance on my dissertation research. Dr. H. Quynh Dinh for introducing me to radial basis functions, graphics hardware, and research in general, as an undergraduate at Stevens Institute of Technology. The work on Fourier volume rendering was initiated under her advisement. Dr. Greg Slabaugh for his supervision during my internships at Siemens Corporate Research, and his advice during my transition to graduate school. Sumit Gupta and NVIDIA Corporation for providing hardware for development and testing.

# Table of Contents

# List of Tables

# List of Figures

# Abstract

KERNEL-BASED MESHLESS METHODS

Andrew Corrigan, PhD

George Mason University, 2009

Dissertation Co-Directors: Dr. John Wallin, Dr. Thomas Wanner

In order to improve their applicability as a tool for solving partial differential equations in computational science, we equip kernel-based meshless methods with a number of new capabilities. First, we provide kernel-based meshless methods with the first wellposed, general technique which allows for adaptively-scaled trial functions. This is done by constructing an adaptively-scaled kernel which maintains positive definiteness. We extend sampling inequalities to optimally bound fractional order Sobolev norms in terms of possibly higher order data. This sampling inequality is then applied to obtain more optimal error bounds in a reformulation of Schaback's framework for unsymmetric meshless methods. We provide kernel-based meshless methods with a direct visualization technique, by adapting Fourier volume rendering to deal directly with meshless data, which was previously only used directly for grid-based data. Modern graphics hardware has emerged as a powerful architecture for scientific computing. We implement an unstructured grid-based inviscid, compressible flow solver on modern graphics hardware, and obtain an order of magnitude speed-up in comparison to an equivalent code running on a quad-core CPU.

# Chapter 1: Introduction

## 1.1 Meshless Methods

The most widely used methods in computational science for solving large-scale problems with complex domain geometries are finite-element and other mesh-based methods. The generation of a mesh for these methods is a bottleneck for many applications in computational science. Not only is mesh generation computationally expensive, it is not robust, and sometimes requires manual user intervention. Meshless methods [5,24,62,75] avoid the need for a mesh, and only require a finite set of points in order to discretize a domain. In contrast with mesh generation, point generation has been shown to be an order of magnitude more efficient than mesh generation and significantly more robust [43]. If meshless methods are otherwise as capable as finite-element methods, meshless methods have the potential to succeed finite-element methods as the main tool for solving partial differential equations in computational science. In this dissertation, we will attempt to provide meshless methods with capabilities that they currently lack in comparison to finite-element methods, which would greatly improve their applicability to computational science.

The field of meshless methods is surveyed in [62, Section 12]. There are a number of different meshless methods, which can roughly be categorized based on the underlying approximation used. Meshless methods based on moving least squares approximation [62, Section 7.9], denoted MLS, are particularly popular in the engineering community [5]. Moving least squares approximation, in the mathematical community, dates back to the work of Lancaster and Salkauskas [40]. The earliest variant of these methods is *smoothed particle hydrodynamics* of Monaghan [49], which employ a convolution-type kernel-based approximation [62, Section 7.2] (a degenerate case of MLS which does not enforce polynomial

reproduction). The value of a moving least squares approximation at a given point is implicitly defined by a local, weighted least squares problem [62, Definition 7.15], generally using polynomials. A notable feature of moving least squares approximation is the requirement that at each evaluation point a small linear system of equations must be constructed and solved. Because of the expense and complexity of sampling such an implicitly defined approximation, particularly when performing quadrature [4] and visualization [16], these methods are not considered here.

We instead turn to meshless methods based on kernel-based interpolation [62, Section 7.5]. These methods form explicit basis or trial functions by centering a kernel over a finite subset of a domain. Kernel-based interpolation originated in the 1970s with the multiquadrics of Hardy [33], and thin-plate splines of Duchon [19]. Multiquadrics and thin-plate splines were popularized due to a favorable comparison with other multivariate interpolation techniques by Franke [27] in 1982, who also conjectured their well-posedness. Both of these methods turn out to share the property of employing conditionally positive definite radial basis functions [75, Chapter 8], which was used by Micchelli [48] in 1986 to establish their well-posedness. Conditional positive definiteness is a generalization of the notion of positive definiteness stated in Definition 1.2 which allows for well-posed interpolation by augmenting the interpolation matrix with additional rows and columns and assuming certain conditions on the point distribution. We only mention this generalization for historical purposes, and do not discuss it further since we are only interested in compactly-supported kernels, for which conditional positive definiteness implies positive definiteness [75, Theorem 9.1]. Both multiquadrics and thin-plate splines are globally-supported, which results in dense interpolation matrices. During the 1990s Schaback [60] and Wendland [74] constructed compactly-supported radial basis functions, which lead to sparse interpolation matrices. These compactly-supported radial basis functions are also positive definite and can be used for well-posed kernel-based interpolation. The field has since greatly matured with a number of textbooks providing a complete overview: Buhmann [11], Wendland [75], and Fasshauer [24]. While these methods have shown to be very promising, a number of

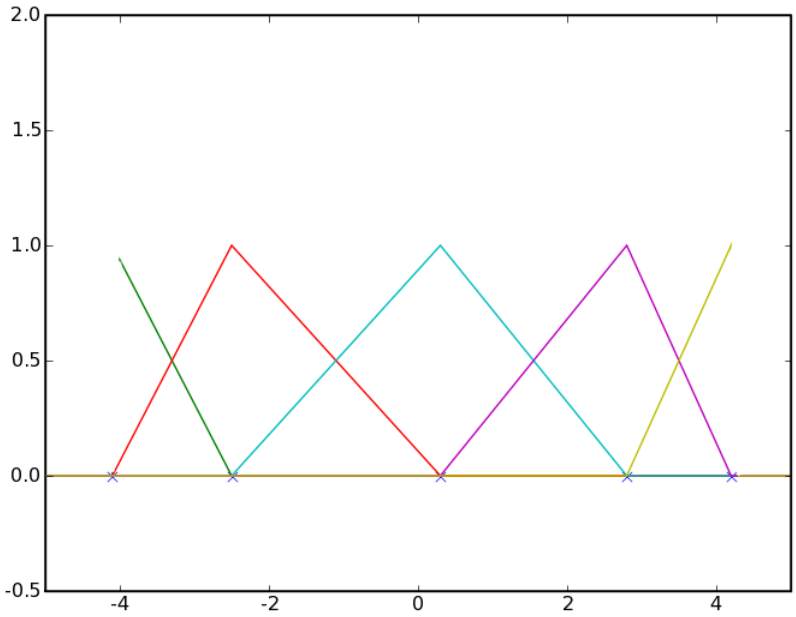



Figure 1.1: A finite-element trial space over the domain $(-4, 4)$. Finite element trial spaces inherently allow for an adaptive scaling of the trial functions which can account for local variations in the density of the point distribution.

important issues remain open in both their mathematical theory and the status of their practical capabilities. Before discussing theses issues, we first give a formulation of well-posed interpolation using kernels, as well as a discussion of the fundamental issue of error estimates in Sobolev spaces.

## 1.2 Kernel-Based Interpolation

Let $\Omega$ be a domain, i.e., an open and bounded subset, in $\mathbb{R}^n$. Given data values $u_1 \dots u_N \in \mathbb{R}$ at a finite set of points $X = \{x_1 \dots x_N\} \subset \Omega$, we consider the problem of finding an *interpolant* $u_{V,X} \in V$ such that

$$u_{V,X}(x_i) = u_i \text{ for } i = 1 \dots N. \tag{1.1}$$

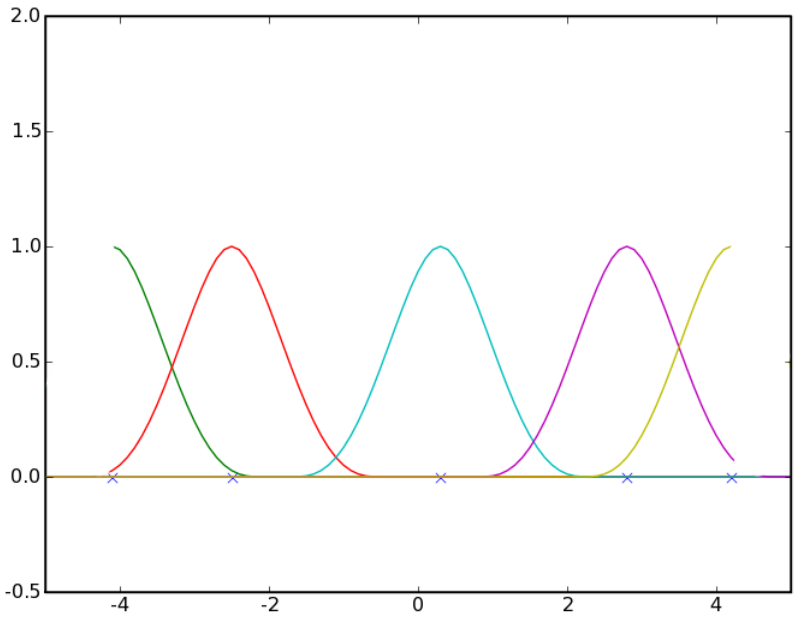



Figure 1.2: A kernel-based trial space over the domain $(-4, 4)$. Kernel-based trial spaces currently are only well-posed in general using a uniform scaling of the trial functions, even if the underlying point distribution has local variations in density.

where $V$ is a finite dimensional *trial space*. The first step to solving this problem for arbitrary data $u$ is to specify the trial space $V$ by choosing basis functions $v_1 \dots v_N : \Omega \to \mathbb{R}$, which are also known as *trial functions*. This allows for the interpolant to be uniquely specified in terms of coefficients $\alpha_1 \dots \alpha_N$,

$$u_{V,X} = \sum_{k=1}^{N} \alpha_k v_k. \tag{1.2}$$

Combining (1.1) and (1.2) results in a linear system of equations, associated with the interpolation matrix

$$A_{V,X} := \left(v_j\left(x_i\right)\right)_{1 \le i,j \le N}. \tag{1.3}$$

For example, finite element interpolation uses piecewise polynomials defined over a mesh as trial functions. This is illustrated in one dimension using piecewise linear polynomials in Figure 1.1. Here we are concerned with the case that the basis functions are defined by centering a kernel, i.e. a bivariate mapping $K : \Omega \times \Omega \to \mathbb{R}$ with $\Omega \subseteq \mathbb{R}^n$, at a finite set of points $X = \{x_1 \dots x_N\}$ in space.

$$v_k := K\left(\cdot, x_k\right) \text{ for } k = 1 \dots N \tag{1.4}$$

An example of such a kernel-based trial space is illustrated in Figure 1.2. Since the choice of kernel and point distribution determines the trial space, we will sometimes write the interpolant as $u_{K,X}$. Currently, the most widely used kernels are undoubtedly the translation-invariant kernels.

**Definition 1.1.** A kernel $K : \Omega \times \Omega \to \mathbb{R}$ is translation-invariant if and only if there exists a function $\Phi : \Omega - \Omega \to \mathbb{R}$ such that

$$K\left(x, y\right) := \Phi\left(x - y\right) \text{ for all } x, y \in \Omega.$$

In this case, the interpolant can be written in the form

$$u_{\Phi,X} = \sum_{k=1}^{N} \alpha_k \Phi\left(\cdot - x_k\right), \tag{1.5}$$

and its interpolation matrix can be denoted $A_{\Phi,X}$. A particular class of functions can be used to construct translation-invariant kernels with invertible interpolation matrices (1.3).

**Definition 1.2.** [75, Definition 6.1] A continuous function $\Phi : \mathbb{R}^n \to \mathbb{R}$ is positive definite if for all $N \in \mathbb{N}$ and all sets of pairwise distinct centers $X = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^n$, the interpolation matrix $A_{\Phi,X}$ is positive definite.

**Remark 1.1.** We follow the naming scheme introduced by Wendland [75], in which "positive definite functions" are required to have positive definite interpolation matrices. Historically, "positive definite functions" were only required to have positive semi-definite interpolation matrices.

Since positive definite matrices are invertible, this definition makes it clear that interpolation using positive definite functions is well-posed, but provides no practical means for the characterization of such functions. Within the class of integrable functions, which are denoted by $L^1\left(\mathbb{R}^n\right)$, an alternative characterization is available which provides such a means.

**Theorem 1.1.** [75, Theorem 6.11] A continuous and integrable function $\Phi$ is positive definite if and only if $\Phi$ is bounded and its Fourier transform is nonnegative and not zero everywhere.

It should be noted that this characterization is a consequence of Bochner's famous characterization of positive *semi*-definite functions [75, Theorem 6.6]. A related result provides a means for constructing positive definite functions.

**Theorem 1.2.** [75, Corollary 6.9] If $f$ is continuous, integrable, nonnegative, and not zero everywhere then its Fourier transform is positive definite.

Most commonly, $\Phi$ is a radial function, i.e., $\Phi$ is defined in terms of a univariate function $\phi : \mathbb{R}^+ \to \mathbb{R}$ such that $\Phi := \phi(\|\cdot\|_2)$. This special case of kernel-based interpolation coincides with the relatively popular technique known as *radial basis function interpolation.*

## 1.3 Error Estimates in Sobolev Space

Computational scientists must always make a tradeoff between the accuracy of a simulation and the amount of computational effort required to perform the simulation. For a physical simulation to be meaningful, the governing equations must be solved with sufficient accuracy, but such accuracy requires more of limited computational resources. Therefore, error estimates, which provide a means for quantifying the relationship between accuracy and effort, are an essential tool for computational scientists.

This quantification is typically done for functions in Sobolev spaces in terms of Sobolev norms. In order to introduce Sobolev spaces we first introduce Lebesgue spaces. Let $f$ be a measurable function. For $p \in [1, \infty)$, $\|f\|^p_{L^p(\Omega)} := \int_\Omega |f|^p$, while $\|f\|_{L^\infty(\Omega)} := \operatorname{ess\,sup}_{x\in\Omega} |f|$. A function is contained in a Lebesgue space if and only if the corresponding Lebesgue norm is finite. Let $L^1_{loc}(\Omega)$ denote the set of all functions which are integrable over any compact subset of $\Omega$. Let $u, v \in L^1_{loc}(\Omega)$ and $\alpha \in \mathbb{N}^n_0$ be a multi-index, in other words, an $n$-tuple of nonnegative integers. The order of the multi-index is denoted $|\alpha|$ and is the sum of each of its components. Suppose that $v$ satisfies

$$\int_\Omega u \partial^\alpha \phi = (-1)^{|\alpha|} \int_\Omega v\phi$$

for all test functions $\phi$ over $\Omega$, i.e., those functions which are infinitely smooth in $\Omega$ with support contained in a compact subset of $\Omega$. Under this assumption $v$ is called the weak partial derivative of order $\alpha$ of $u$, which is denoted by $\partial^\alpha u$. Let $r \in \mathbb{N}$ and $p \in [1, \infty]$, the integer order Sobolev space $W^{r,p}(\Omega)$ consists of all functions $f \in L^p(\Omega)$ whose weak partial derivatives for all multi-indexes $\alpha$ such that $|\alpha| \leq$ exist and are contained in $L^p(\Omega)$. In the

case that $p \in [1, \infty)$, it is equipped with the semi-norm

$$|f|^p_{W^{r,p}(\Omega)} := \sum_{|\alpha|=r} |\partial^\alpha f|^p_{L^p(\Omega)}\,, \tag{1.6}$$

and norm

$$\|f\|^p_{W^{r,p}(\Omega)} := \sum_{|\alpha|\leq r} |\partial^\alpha f|^p_{L^p(\Omega)}\,, \tag{1.7}$$

while in the case that $p = \infty$

$$|f|_{W^{r,\infty}(\Omega)} := \max_{|\alpha|=r} |\partial^\alpha f|_{L^\infty(\Omega)} \tag{1.8}$$

and

$$\|f\|_{W^{r,\infty}(\Omega)} := \max_{|\alpha|\leq r} |\partial^\alpha f|_{L^\infty(\Omega)}\,. \tag{1.9}$$

If $r$ is a positive real number such that $r \notin \mathbb{N}$ then the fractional order Sobolev space $W^{r,p}(\Omega)$ consists of all functions $f \in W^{\lfloor r \rfloor, p}(\Omega)$ for which the semi-norm $|f|_{W^{r,p}(\Omega)}$ is finite. Here $\lfloor r \rfloor$ denotes the largest integer $k$ such that $k \leq r$, while $\lceil r \rceil$ denotes the smallest integer $k$ such that $k \geq r$. In the case that $p \in [1, \infty)$ this semi-norm is defined by

$$|f|^p_{W^{r,p}(\Omega)} := \int_\Omega \int_\Omega \frac{|f(x) - f(y)|^p}{|x-y|^{n+(r-\lfloor r \rfloor)p}} dxdy, \tag{1.10}$$

while in the case that $p = \infty$ it is defined by

$$|f|_{W^{r,\infty}(\Omega)} := \operatorname*{ess\,sup}_{x,y\in\Omega, x\neq y} \frac{|f(x) - f(y)|}{|x-y|^{r-\lfloor r \rfloor}}. \tag{1.11}$$

For $p \in [1, \infty)$, the fractional order Sobolev space is equipped with the norm defined by

$$\|f\|^p_{W^{r,p}(\Omega)} := \|f\|^p_{W^{\lfloor r \rfloor, p}(\Omega)} + |f|^p_{W^{r,p}(\Omega)}, \tag{1.12}$$

while in the case $p = \infty$ it is equipped with the norm defined by

$$\|f\|_{W^{r,\infty}(\Omega)} := \max\left(\|f\|_{W^{\lfloor r \rfloor, \infty}(\Omega)}, |f|_{W^{r,\infty}(\Omega)}\right). \tag{1.13}$$

We employ the abbreviations $|f|_{r,p,\Omega} := |f|_{W^{r,p}(\Omega)}$ and $\|f\|_{r,p,\Omega} := \|f\|_{W^{r,p}(\Omega)}$. Of particular importance are Sobolev spaces with $p = 2$, which are denoted by $H^r(\Omega)$. It is possible to equip $H^r(\mathbb{R}^n)$ with a norm which is equivalent to either (1.7) if $r \in \mathbb{N}$, or (1.12) if $r \notin \mathbb{N}$,

$$\|f\|_{H^r(\mathbb{R}^n)} := \left\| \mathcal{F}_n f(\cdot) \left(1 + \|\cdot\|_2^2\right)^{r/2} \right\|_{L^2(\mathbb{R}^n)} \tag{1.14}$$

where $\mathcal{F}_n f$ denotes the Fourier transform of $f$

$$\mathcal{F}_n f(\xi) := \int_{\mathbb{R}^n} f(x) e^{-ix^T \xi} dx. \tag{1.15}$$

Throughout this work we will usually assume that $\Omega$ has a Lipschitz-continuous boundary. A Lipschitz continuous boundary is one that can locally be represented as the graph of a Lipschitz continuous function. This assumption will allow for the application of Sobolev embedding and extension theorems. A weaker version of a Sobolev embedding theorem stated by Arcangéli et al. [2] is stated as follows.

**Theorem 1.3.** [2, Proposition 2.1] Let $\Omega$ be a bounded domain in $\mathbb{R}^n$ with a Lipschitz-continuous boundary. Then, for any $p \in [1, \infty]$, nonnegative integer $\mu$, and real number $r > n/p$,

$$W^{r,p}(\Omega) \hookrightarrow C^{\mu}\left(\overline{\Omega}\right).$$

The following is a restatement of the existence of an extension operator, based on that of Arcangéli et al. [2].

**Theorem 1.4.** [2, Eq. 2.3] Let $\Omega$ be a bounded domain in $\mathbb{R}^n$ with a Lipschitz-continuous boundary. Then, for any $p \in [1, \infty)$ and $r \geq 0$, there exists a linear continuous operator $E : W^{r,p}(\Omega) \to W^{r,p}(\mathbb{R}^n)$ such that, for any $v \in W^{r,p}(\Omega)$, $Ev|_\Omega = v$. Moreover, such an operator $E$ also exists if $p = \infty$ and $r \in \mathbb{N}$.

The solutions of the partial differential equations of computational science are typically contained in a Sobolev space $H^r(\Omega)$ where $r \geq 0$. Let $0 \leq l \leq r$, then a Sobolev error estimate is of the form

$$\|u - u_h\|_{H^l(\Omega)} \leq \epsilon(h) \|u\|_{H^r(\Omega)} \text{ for all } u \in H^r(\Omega), \tag{1.16}$$

where the optimal form of the *error factor* is

$$\epsilon(h) = Ch^{r-l}. \tag{1.17}$$

The *error* is the distance of a function $u \in U$ to its approximation $u_h$ measured in the norm $\|\cdot\|_{H^l(\Omega)}$. The order $l$ is the maximum order of derivatives controlled by the error estimate. For example, if $l = 1$, then the error estimate will control up to first order derivatives of the solution. As can be seen from (1.17), there is a tradeoff between the order of convergence and the order of the Sobolev norm in which the convergence takes places.

The error is then bounded by $\epsilon(h)$, the *error factor,* scaled by the size of the function $u$ measured in the norm $\|\cdot\|_{H^r(\Omega)}$. The error factor $\epsilon(h)$ depends on the *discretization parameter* $h$ which typically measures how dense the domain discretization is in the domain, and is directly related to the computational effort. In the context of meshless methods the

discretization parameter is the *fill distance*

$$h\left(X,\Omega\right) := \sup_{y\in\Omega}\min_{x\in X}\left\|x-y\right\|, \tag{1.18}$$

which measures the radius of the largest open ball in the domain which does not contain a discretization point. Since the error estimate in (1.16) includes the unknown solution in the estimate, it cannot explicitly quantify the error. It is still of practical value since it can instead be used to describe how quickly one should expect the error to decrease, which is mathematically described by the notion of the rate of convergence. For example, the optimal error factor (1.17), which behaves asymptotically like $h^{r-l}$, is said to converge with order $r-l$. Such a property implies useful statements such as, "if twice the computational effort is exerted, then it follows that the accuracy in the $l$th order derivatives will improve by a factor of $2^{r-l}$".

## 1.4 Native spaces

For their application in Chapter 2, we introduce relevant facts regarding the native spaces associated with a class of translation-invariant kernels. A full development of the theory of native spaces is provided by Wendland [75, Chapter 10]. Suppose that $K$ is translation-invariant, and thus defined in terms of a function, denoted $\Phi : \mathbb{R}^n \to \mathbb{R}$, which is positive definite on $\mathbb{R}^n$ and whose Fourier transform satisfies,

$$C_2^{-1}\left(1+\left\|\omega\right\|_2^2\right)^r \leq \mathcal{F}_n\Phi\left(\omega\right)^{-1} \leq C_1^{-1}\left(1+\left\|\omega\right\|_2^2\right)^r \text{ for all } \omega\in\mathbb{R}^n, \tag{1.19}$$

where the constants $C_1, C_2$ are independent of $\omega$ and satisfy $0 < C_1 \leq C_2$. The native space $\mathcal{N}_\Phi\left(\Omega\right)$ of the kernel $K = \Phi\left(\cdot - \cdot\right)$ is a Hilbert space of functions defined as the completion of the space

$$F_\Phi\left(\Omega\right) := \operatorname{span}\left\{\Phi\left(\cdot - y\right) : y\in\Omega\right\}.$$

It is important to note that the space $F_\Phi(\Omega)$ contains all possible interpolants, and also that the kernel is the reproducing kernel, c.f., Aronszajn [3], of its native space. The native space of this translation-invariant kernel is equipped with the norm

$$\|f\|^2_{\mathcal{N}_\Phi(\mathbb{R}^n)} = \int_{\mathbb{R}^n} \frac{|\mathcal{F}_n f(\omega)|^2}{\mathcal{F}_n \Phi(\omega)} d\omega \text{ for all } f \in \mathcal{N}_\Phi(\mathbb{R}^n)$$

Since there is the alternative characterization of Sobolev space,

$$H^r(\mathbb{R}^n) = \left\{ f \in L^2(\mathbb{R}^n) : \mathcal{F}f(\cdot) \left(1 + \|\cdot\|_2^2\right)^{r/2} \in L^2(\mathbb{R}^n) \right\},$$

it follows from (1.19) that $\mathcal{N}_\Phi(\mathbb{R}^n)$ and $H^r(\mathbb{R}^n)$ consist of the same functions and have equivalent norms:

$$C_2^{-1/2} \|f\|_{H^r(\mathbb{R}^n)} \leq \|f\|_{\mathcal{N}_\Phi(\mathbb{R}^n)} \leq C_1^{-1/2} \|f\|_{H^r(\mathbb{R}^n)} \text{ for all } f \in H^r(\mathbb{R}^n).$$

Since $K$ is a positive definite kernel we have from [75, Corollary 10.25] that the following bound holds in the kernel's native space,

$$\|u_{\Phi,X} - u\|_{\mathcal{N}_\Phi(\mathbb{R}^n)} \leq \|u\|_{\mathcal{N}_\Phi(\mathbb{R}^n)} \text{ for all } u \in \mathcal{N}_\Phi(\mathbb{R}^n). \tag{1.20}$$

## 1.5 Data-Dependent and Irregular Memory Access Patterns

In high-performance computing a crucial issue that arises is the pattern in which memory is accessed. Traditional CPU-based architectures heavily rely on caches, which are extremely efficient to access compared to the main system memory since they are located on-chip. When software accesses data stored in the main system memory, rather than just transferring the individual piece of data which was requested, the CPU may also decide to read in data stored in other nearby locations in main system memory. Thus, if successive memory

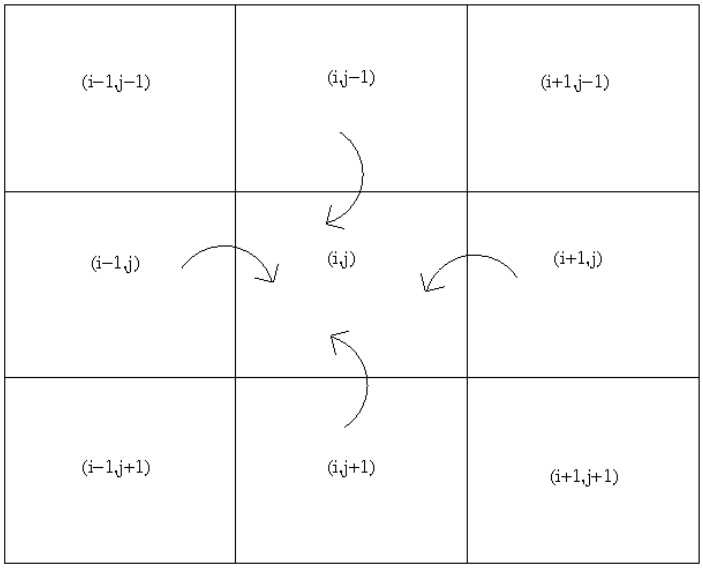



Figure 1.3: An example of the fixed memory access pattern of structured grid based solvers. Within a loop over the elements, data stored at four neighboring elements are read, and the array indexes of this data are known in advance and follow a fixed pattern.

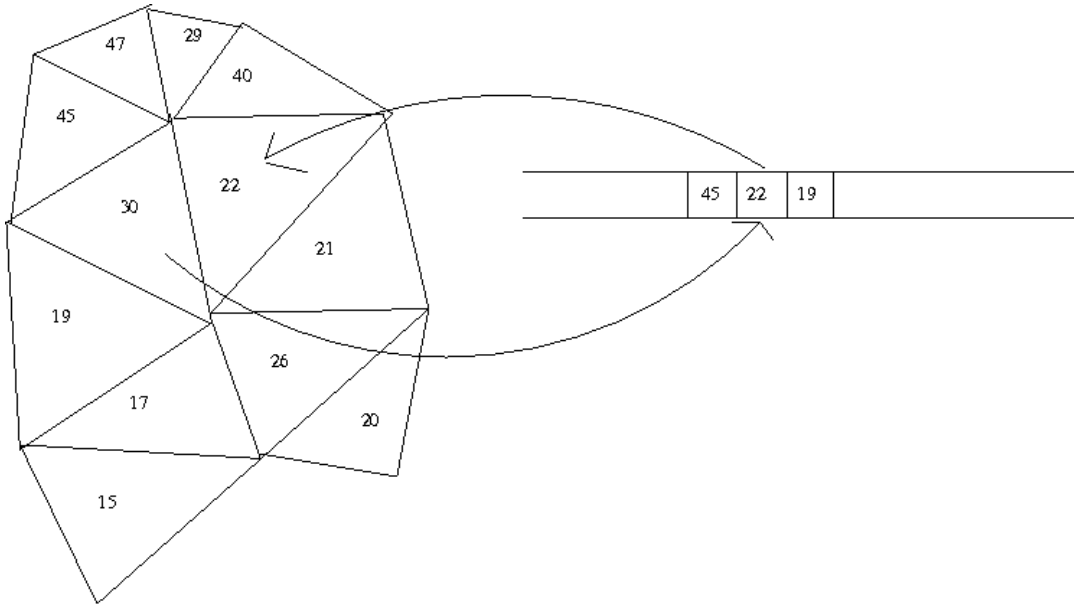



Figure 1.4: An example of the data-dependent memory access pattern of unstructured grid based solvers. Within a loop over the elements, data stored at neighboring elements are read, but the array indexes of neighbors must first be read from a connectivity array, and obey no specific pattern.

accesses are to these nearby locations, then the main system memory will not need to be read from again. This is usually called a *cache hit*. However, if successive memory accesses are to distant locations in memory, which have not already been read into the on-chip cache, then an entirely new chunk of memory will be read in from main system memory. This is known as a *cache miss*. A memory access pattern which leads primarily to cache hits, will achieve a relatively high level of performance, or at least any lack of performance will be due to other factors. When a memory access pattern is used which leads primarily to cache misses, a large degradation of performance can be expected, since reading from system memory is much slower, sometimes even an order of magnitude or more.

For structured grid based solvers, obtaining a good memory access pattern is trivial. When a loop is performed over the elements of a structured grid, each element will access its neighbors using indexes which are known a priori and in a fixed pattern. This is illustrated in Figure 1.3. For unstructured grid based solvers, the neighbors are not accessed based on such a fixed pattern. Before accessing neighbors, the indexes of the neighbors must first be read from a connectivity array. This is called *indirect* or *data-dependent memory access*. The flux calculation code given in Appendix B illustrates this. Furthermore, once the data is read, the indexes may not exhibit any particular pattern. It is therefore up to *numbering schemes* to ensure that the element indexes are in such an order that nearby elements are as close as possible in memory. On CPU-based architectures this will help ensure that a relatively high ratio of cache hits to misses, and thus relatively high performance. A particular example of this is illustrated in Figure 1.4.

In kernel-based meshless methods, the neighbors of a given trial function center are defined as points which fall within its support. Each row of the interpolation matrix corresponds to a given interpolation point, so that these neighbors correspond to the row's nonzero entries. The indexes of these neighbors can be stored in a connectivity array just like the indexes of neighboring elements are stored in unstructured grid solvers. Although different arithmetic performed, these neighbors are accessed in a data-dependent manner when constructing the sparse interpolation matrix. Unlike structured grid based solvers

and like unstructured grid based solvers, for meshless methods, the indexes stored in the connectivity array will not exhibit any particular pattern and therefore numbering schemes again play an important role in achieving a memory access pattern which leads to high performance.

## 1.6 Graphics Hardware

Modern graphics hardware has emerged as an extremely powerful architecture for scientific computing. For example, the latest NVIDIA GeForce 200 series and NVIDIA Tesla 10 series GPUs now achieve roughly one teraflop of performance, which is an order of magnitude higher performance than high-end CPUs [53, Sec. 1.2]. However, the architecture is fundamentally different than existing architectures and requires new approaches to fully exploit its available computational resources.

We first give a basic introduction of programming modern graphics hardware with CUDA [53] by considering the example of performing element-wise addition of two arrays and storing the result in a third array. The first part of Figure 1.5 shows this in C++/OpenMP. Prior to the loop over each element of the arrays, there is an OpenMP compiler directive specifying that the loop should be parallelized and run in separate threads. On a multicore CPU, these threads can be executed in parallel by each core. The second part of Figure 1.5 shows the equivalent CUDA code. The CUDA kernel, distinguished by the prefix __global__, is executed once for each thread in a grid of thread blocks. The grid of thread blocks is specified when the kernel is invoked via the thread block configuration $\langle\langle\langle$Dg,Db$\rangle\rangle\rangle$, which in this example means that the CUDA kernel is executed over N threads. The block structure organize the threads into groups, where the number of thread blocks is specified by Dg, while the number of threads per block is specified by Db.

At the hardware level, multiple threads are executed in parallel. Each collection of threads executed in parallel is known as a warp. A warp consists of 32 threads, and is divided into half-warps of 16 threads. Memory access is performed in segments for the threads of a half-warp, as opposed to being performed separately for each thread. In the

```
//OpenMP
void openmp_add(int N, float* a, float* b, float* c)
{
  #pragma omp parallel for
  for(int i = 0; i < N; i++)
    c[i] = a[i] + b[i];
}
openmp_add(N, a, b, c);

// CUDA
__global__ void cuda_add(float* a, float* b, float* c)
{
  int i = blockDim.x*blockIdx.x + threadIdx.x;
  c[i] = a[i] + b[i];
}
dim3 Dg(N/block_length), Db(block_length);
cuda_add<<<Dg, Db>>>(a, b, c);
```

Figure 1.5: A comparson of adding two vectors together in parallel using OpenMP on a multicore CPU with adding two vectors together in parallel in using CUDA on NVIDIA graphics hardware.

case of adding two arrays together, for example, a[0...15] is read in one segment by the first sixteen threads, while c[128...143] is written to in one segment by threads 128-143. If threads within a half-warp read from disparate locations in memory then multiple segments must be read and performance can degrade by nearly an order of magnitude. Furthermore, graphics hardware has limited caching facilities, so that any unused data in the memory segments will not be cached for possible later use in the execution of the CUDA kernel. Unlike in the example given in Figure 1.5, in the finite-volume flux calculation in Chapter 5, data is accessed in an unstructured, data-dependent manner based on a connectivity array. The numbering of the elements in the connectivity array, which determines the degree of coalescing, can therefore play a pivotal role in obtaining high performance.

## 1.7 Open Issues

In this dissertation I will consider four problems in meshless methods.

### 1.7.1 Well-Posed Interpolation Using Adaptively-Scaled Kernels

One of the main benefits of kernel-based interpolation is the ability to approximate data at a scattered set of points, without any meshing requirements. However, for kernel-based interpolation to be well-posed it is currently required that the trial functions are scaled uniformly, despite any local variations in the density of the underlying point distributions (see Figure 1.2). In contrast, finite-element interpolation, which can also interpolate data at a scattered set of points as long as a mesh is provided, is formulated in such a way that allows for adaptively-scaled trial functions (see Figure 1.1). The mathematical theory of finite element methods then allows for adaptive error estimates and also more favorable stability results [10, Chapter 9] in comparison to kernel-based interpolation [75, Chapter 6]. Counterexamples have been provided which preclude the well-posedness of an existing approach to constructing adaptively-scaled kernel-based trial spaces (see Figure 2.1) which uniformly scales each trial function within its support. However, we will propose an alternative approach which achieves adaptivity by transforming the underlying domain, regardless of the placement of trial function centers, so that, for example, trial functions may be scaled non-uniformly within their support (see Figure 2.4a). This alternative approach is well-posed and an appropriate adaptivity transformation can be used to improve stability for less uniform point distributions.

### 1.7.2 A Sampling Inequality for Fractional Order Sobolev Semi-Norms Using Arbitrary Order Data

The application of kernel-based meshless trial spaces to the numerical solution of partial differential equations traces back to the pioneering work of Kansa [39]. Kansa modified the interpolation matrix (1.3) to enforce a partial differential equation pointwise. The well-posedness of the resulting collocation matrix remained an open question until the counterexample of Hon and Schaback [36]. To work around this problem, later work by Schaback [63, 64] modified Kansa's original formulation, and provides an abstract framework which can be used to establish quantitative error estimates for kernel-based meshless

methods for partial differential equations. These estimates are currently suboptimal for problems specified over Sobolev spaces, in particular inhomogeneous boundary value problems, due to a lack of sampling inequalities, c.f. (3.24), optimally bounding fractional-order Sobolev norms. To ensure optimal bounds for those problems, we will further generalize existing sampling inequalities to optimally bound fractional-order Sobolev norms. Schaback's framework introduces the notion of a uniformly stable test discretization, without which the order of convergence is diminished. We introduce a new sampling technique in an attempt to obtain such a discretization, and study its effect on the convergence results obtained via Schaback's framework.

### 1.7.3 Visualization Using Fourier Volume Rendering

Volume rendering is an important visualization tool used by computational scientists to extract information from three-dimensional simulation data. In the past, volume rendering techniques primarily focused on data consisting of samples of a function over structured or unstructured grids, or even just at a scattered set of points. This sample-based approach to volume rendering techniques discards the specific form of kernel-based meshless data, which is given by (1.2) and (1.4). There is a relatively limited body of work that deals with meshless data *directly,* i.e., without first sampling the data over a grid, which also accounts for the specific form of kernel-based meshless data (1.2). An indirect approach to visualizing meshless data would be to first sample the meshless data over a grid, and to then use a grid-based visualization technique. This approach treats kernel-based meshless data, a function defined over $\mathbb{R}^3$, as something it is not: samples at a finite subset of $\mathbb{R}^3$. It has been shown that this approach could lead to a loss of important detail at feasible grid resolutions [14, 37, 55, 56]. Furthermore such an approach leaves meshless data inherently less efficient to visualize than pre-sampled grid data. Instead, we adapt a volume rendering technique to produce images directly from kernel-based meshless simulation data.

### 1.7.4 Running Unstructured Grid CFD Solvers on Modern Graphics Hardware

As described in Section 1.6, modern graphics hardware is an extremely powerful architecture for scientific computing. However, the architecture is fundamentally different than existing architectures and requires new approaches to fully exploit its available computational resources. We consider issues which arise in the implementation of an inviscid, compressible flow solver on modern graphics hardware. Since an unstructured grid is used, which results in irregular and data-dependent memory access, the main issue addressed is memory bandwidth. This issue will also be critical in efficient implementations of meshless methods, which require similar irregular and data-dependent memory access.

# Chapter 2: Well-Posed Interpolation Using Adaptively-Scaled Kernels

## 2.1 Introduction

In the classical formulation of radial basis function interpolation, c.f. (1.5), the support sizes of the trial functions are uniform. It is common using other types of multivariate interpolation, such as finite element interpolation, to allow for trial functions with nonuniform support sizes. This is of particular importance when the underlying point distribution has a spatially-varying density. Figure 2.2 illustrates the dilemma of using a fixed kernel size with such a point distribution: a kernel size must be chosen that is relatively large in one region or relatively small in another. However, with an adaptive scaling it may be possible to account for such local variations in the density of the point distribution. To apply the idea of an adaptive scaling to radial basis function interpolation, one approach has been to directly modify the support of each trial function: given kernel sizes $r_1, \ldots, r_N > 0$ then $u_{\Phi,X}$ (1.5) becomes

$$u_{\Phi,X,R} = \sum_{k=1}^{N} \alpha_k \Phi\left(\frac{\cdot - x_k}{r_k}\right). \tag{2.1}$$

An example of a trial space defined in this way is illustrated in Figure 2.1. The advantage of this approach is that roughly the same number of points are contained in the support of each trial function. Unfortunately, the positive definiteness of the associated interpolation matrix $\left(\Phi\left(\frac{x_i - x_j}{r_j}\right)\right)_{1 \le i,j \le N}$ does not follow immediately from the positive definiteness of $\Phi$. Thus, when using an adaptively-scaled interpolant (2.1) the invertibility of the interpolation matrix is not guaranteed. In fact there are known cases of singular interpolation matrices,

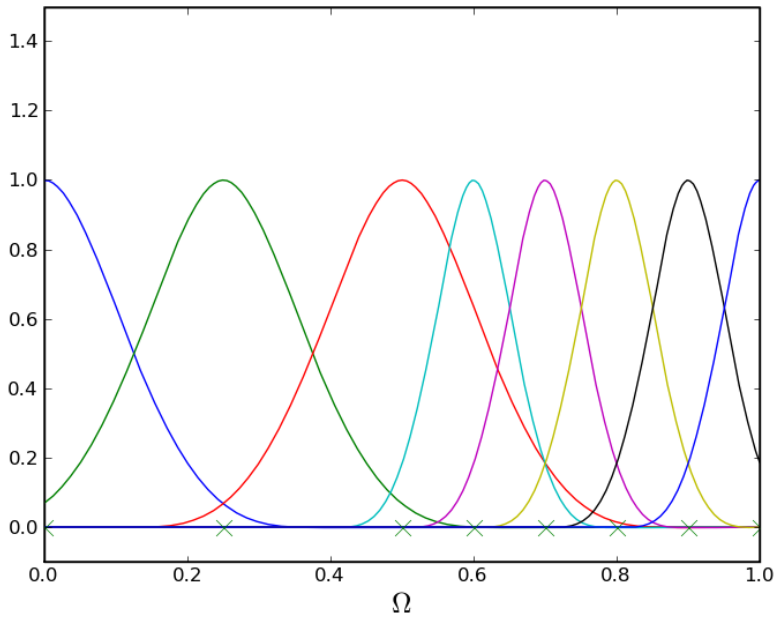
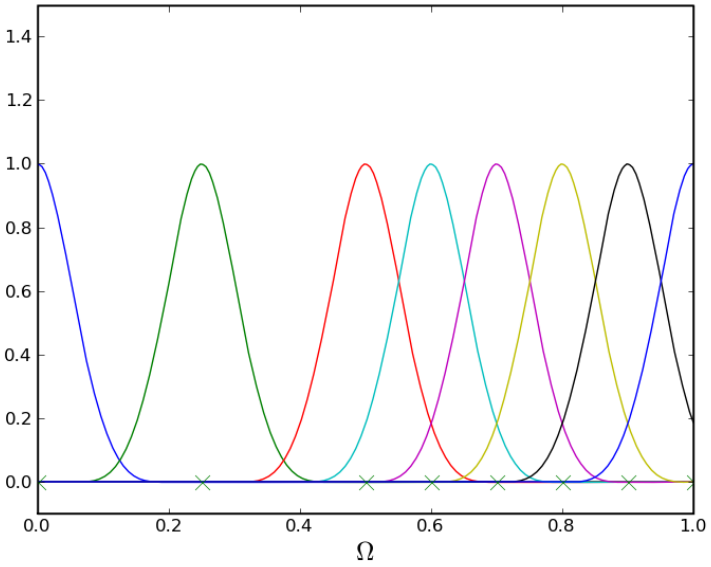



Figure 2.1: An adaptive scaling where each trial function is possibly scaled differently, as per (2.1). This approach is not well-posed in general. The condition number of the associated interpolation matrix is 2.024.

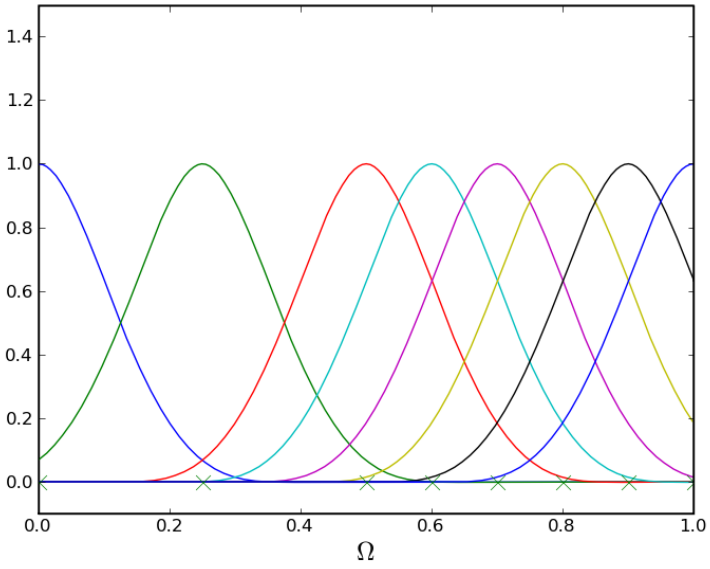
given by both Buhmann [11, Page 154] and Fornberg and Zuev [25, Section 4.2]. Furthermore, Schaback [66] has demonstrated the possibility of singularity numerically, analogous to previous work with Hon regarding the well-posedness of Kansa's method [36]. Despite this, numerical results, provided by Driscoll and Heryudono [17], Fornberg and Zuev [25], and Wertz et al. [76], have shown that this approach can improve the stability and accuracy of interpolation.

To achieve a well-posed scheme for radial basis function interpolation, using adaptively-scaled trial functions, one may impose additional conditions. This has been done by Bozzini et al. [7] for the particular case of one-dimensional approximate multiquadrics with certain conditions on the scales which ensure the diagonal dominance of the interpolation matrix.

Rather than studying sufficient conditions to achieve the unique existence of the adaptively-scaled interpolant (2.1) for particular radial basis functions, the purpose of this chapter is to consider an alternative approach for achieving adaptivity, which works in a very general




(a) "Too small" of a kernel size has been chosen for trial functions centered in $\Omega_1 = \left(0, \frac{1}{2}\right)$. The condition number of the associated interpolation matrix is 2.021.




(b) "Too large" of a kernel size has been for trial functions centered in $\Omega_2 = \left(\frac{1}{2}, 1\right)$. The separation distance is $\frac{1}{20}$. The condition number of the associated interpolation matrix is 24.933.

Figure 2.2: Interpolation using a nonadaptively-scaled translation-invariant kernel (1.5). A fixed kernel size must be chosen which is either "too small" (2.2a) or "too large" (2.2b).

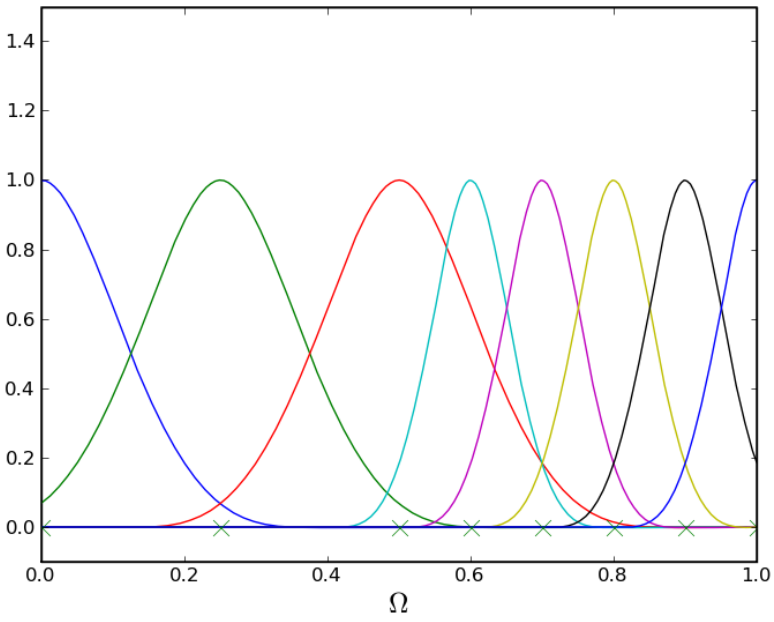



Figure 2.3: An adaptive scaling where each trial function is possibly scaled differently, as per (2.1). This approach is not well-posed in general. The condition number of the associated interpolation matrix is 2.030.

situation. This alternative approach, which is formulated in Section 2.2, is well-posed since an adaptive kernel is used which maintains continuity, symmetry, and positive definiteness. In Section 2.3, a basic error estimate is established which demonstrates convergence when using a fixed kernel, i.e., the case of nonstationary refinement. Finally, in Section 2.4 preliminary computational results are provided to illustrate theory.

## 2.2 Formulation and Well-Posedness

Let $\hat{\Omega}, \Omega \subset \mathbb{R}^n$, $T : \hat{\Omega} \to \Omega$ be a bi-Lipschitz homeomorphism, and $\hat{K} : \hat{\Omega} \times \hat{\Omega} \to \mathbb{R}$ be a continuous, symmetric, positive definite kernel, e.g., a radial basis function. By bi-Lipschitz it is meant that both $T$ and its inverse $T^{-1}$ are Lipschitz mappings. It is assumed that $\Omega$ is a Lipschitz domain, i.e., $\Omega$ is open, bounded, and has a Lipschitz continuous boundary. Due to the conditions on $T$, it follows that $\hat{\Omega}$ is also a Lipschitz domain. It follows that the domain $\hat{\Omega}$ satisfies the cone property [2, Page 185] with radius $\hat{\rho} > 0$ and angle $\hat{\theta} \in (0, \pi/2]$. Recall that the kernel $\hat{K} : \hat{\Omega} \times \hat{\Omega} \to \mathbb{R}$ is *positive definite* if and only if for an arbitrary finite subset $\hat{X} = \{\hat{x}_1, \ldots, \hat{x}_N\}$ of $\hat{\Omega}$ the *interpolation matrix*

$$A_{\hat{K},\hat{X}} := \left(\hat{K}\left(\hat{x}_i, \hat{x}_j\right)\right)_{1 \le i,j \le N}$$

is positive definite, and thus invertible. We define an *adaptively scaled kernel*

$$K(x, y) := \hat{K}\left(T^{-1}x, T^{-1}y\right) \text{ for all } (x, y) \in \Omega \times \Omega. \tag{2.2}$$

**Proposition 1.** *The adaptively-scaled kernel* (2.2) *is continuous, symmetric, and positive definite under the above stated conditions. Furthermore the interpolants are related via*

$$u_{K,X} = \hat{u}_{\hat{K},\hat{X}} \circ T^{-1}. \tag{2.3}$$

*Proof.* Continuity and symmetry follow immediately from the properties of $K$ and $T$ stated above . Let $X = \{x_1, \ldots, x_N\}$ be an arbitrary finite subset $X$ of $\Omega$. Let $\hat{X} := T^{-1}X$, a finite subset of $\hat{\Omega}$, which also consists of $N$ distinct points, ensured by the fact that $T$ is invertible. It follows immediately that $A_{K,X}$ and $A_{\hat{K},\hat{X}}$ coincide, and therefore both are positive definite since $\hat{K}$ is positive definite. The relation (2.3) holds since the coefficients of each interpolant coincide, and the trial functions $v_k := K(\cdot, x_k)$ and $\hat{v}_k := \hat{K}(\cdot, \hat{x}_k)$ are related via $v_k = \hat{v}_k \circ T^{-1}$. □

The well-posedness of kernel-based interpolation, as formulated in Section 1.2, using an adaptively-scaled kernel (2.2) is ensured by Proposition 1.

## 2.3 Error Estimates and Stability

In this section, we first review existing theory for error estimates of nonadaptively-scaled translation-invariant kernels. The result provided in this case is then used to establish a basic error estimate in the case of adaptively-scaled kernels. Finally, the effect of the adaptivity transformation on the stability of interpolation is discussed.

### 2.3.1 Error Estimates for Interpolation Using Nonadaptively-Scaled Translation-Invariant Kernels

We introduce the additional condition on $\hat{K}$ that it is translation-invariant, and thus defined in terms of a function, denoted $\hat{\Phi} : \mathbb{R}^n \to \mathbb{R}$, which is positive definite on $\mathbb{R}^n$ and whose Fourier transform satisfies (1.19).

Recently, a powerful technique has emerged for establishing optimal error estimates using so-called *sampling inequalities* which bound a norm in terms of another norm and some samples of the function. The presentation given here of this technique is based on that of Narcowich, et al. [50]. Let $0 \leq l < r$ such that $r - l \in \mathbb{N}$ and $r > n/2$. The following

sampling inequality is Corollary 3 of Chapter 3 with $\mu = 0$,

$$\|\hat{u}\|_{H^l(\hat{\Omega})} \leq C \left( \hat{h}^{r-l} \|\hat{u}\|_{H^r(\hat{\Omega})} + \hat{h}^{n/2-l} \left\| \hat{u}|_{\hat{X}} \right\|_{\ell_2} \right) \text{ for all } \hat{u} \in H^r\left(\hat{\Omega}\right), \tag{2.4}$$

where $\hat{X}$ is an arbitrary finite subset of the domain $\hat{\Omega}$ with corresponding discretization parameter $\hat{h} = h\left(\hat{X}, \hat{\Omega}\right)$, c.f. (1.18). The hypotheses of Corollary 3 imply that C is dependent on $\hat{\Omega}, n, r$, and $l$, and require that $\hat{h} \leq \hat{\mathfrak{d}}_r$, where $\hat{\mathfrak{d}}_r$ is dependent on $\hat{\Omega}, n$, and $r$.

Given data $\hat{u}|_{\hat{X}}$, it follows from Section 1.2 that there exists a unique interpolant $\hat{u}_{\hat{\Phi},\hat{X}} \in V_{\hat{\Phi},\hat{X}}$ satisfying

$$\left( \hat{u}_{\hat{\Phi},\hat{X}} - \hat{u} \right)\Big|_{\hat{X}} = 0. \tag{2.5}$$

**Proposition 2.** *(Narcowich, et al. [50, Lemma 3.1]) We have that*

$$\left\| \hat{u}_{\hat{\Phi},\hat{X}} - \hat{u} \right\|_{H^r(\hat{\Omega})} \leq (C_2/C_1)^{1/2} \|E\| \|\hat{u}\|_{H^r(\hat{\Omega})}, \tag{2.6}$$

*where $E : H^r(\Omega) \to H^r(\mathbb{R}^n)$ is a continuous, linear, extension operator (Theorem 1.4), and the constants $C_1$ and $C_2$ depend on $\hat{\Phi}$.*

*Proof.* From Theorem 1.4 it follows that there exists a continuous, linear, extension operator $E : H^r\left(\hat{\Omega}\right) \to H^r(\mathbb{R}^n)$ satisfying $E\hat{u}|_{\hat{\Omega}} = \hat{u}$, which implies that $\hat{u}_{\hat{\Phi},\hat{X}} = (E\hat{u})_{\hat{\Phi},\hat{X}}\Big|_{\hat{\Omega}}$.

Combining this with (1.19) and (1.20),

$$\begin{aligned}
\left\|\hat{u}_{\hat{\Phi},\hat{X}} - \hat{u}\right\|_{H^r(\hat{\Omega})} &= \left\|(E\hat{u})_{\hat{\Phi},\hat{X}} - E\hat{u}\right\|_{H^r(\hat{\Omega})} \\
&\leq \left\|(E\hat{u})_{\hat{\Phi},\hat{X}} - E\hat{u}\right\|_{H^r(\mathbb{R}^n)} \\
&\leq C_2^{1/2} \left\|(E\hat{u})_{\hat{\Phi},\hat{X}} - E\hat{u}\right\|_{\mathcal{N}_{\hat{\Phi}}(\mathbb{R}^n)} \\
&\leq C_2^{1/2} \left\|E\hat{u}\right\|_{\mathcal{N}_{\hat{\Phi}}(\mathbb{R}^n)} \\
&\leq (C_2/C_1)^{1/2} \left\|E\hat{u}\right\|_{H^r(\mathbb{R}^n)} \\
&\leq (C_2/C_1)^{1/2} \left\|E\right\| \left\|\hat{u}\right\|_{H^r(\hat{\Omega})} .
\end{aligned}$$

□

An error estimate, with the expected order of convergence, then follows from (2.4), (2.5), and (2.6),

$$\left\|\hat{u}_{\hat{\Phi},\hat{X}} - \hat{u}\right\|_{H^l(\hat{\Omega})} \leq C\hat{h}^{r-l} \left\|\hat{u}\right\|_{H^r(\hat{\Omega})} . \tag{2.7}$$

where the constant $C$ inherits all of the dependencies stated above, and $\hat{h}$ is required to be sufficiently small to satisfy the requirement stated above.

### 2.3.2 Error Estimates for Interpolation Using Adaptive Kernels

An error estimate in the domain $\Omega$ using the adaptively-scaled kernel $\Phi$, c.f. (2.2) follows immediately if the following properties are satisfied:

$$u \in H^r(\Omega) \iff \hat{u} \in H^r\left(\hat{\Omega}\right) \tag{2.8}$$

and that for all $u \in H^r(\Omega)$,

$$\|u\|_{H^r(\Omega)} \leq C_1(T) \|\hat{u}\|_{H^r\left(\hat{\Omega}\right)} \leq C_2(T) \|u\|_{H^r(\Omega)}. \tag{2.9}$$

These properties can be established based on a result given by Ciarlet [13, Theorem 4.3.2] which relates the Sobolev semi-norms of functions which have been composed with a sufficiently smooth mapping.

**Theorem 3.** *In addition to the conditions stated in Sections 2.2 and 2.3.1, suppose that $T$ is a transformation satisfying* (2.8) *and* (2.9)*, then if $h(X,\Omega) \leq \mathfrak{d}_r(T,\Omega,n,r)$ we have that*

$$\|u - u_{\Phi,X}\|_{H^l(\Omega)} \leq C \cdot h(X,\Omega)^{r-l} \|u\|_{H^r(\Omega)},$$

*where $C$ is a constant which depends on $\Phi, T, \Omega, n, r,$ and $l$.*

Since the constant in the error estimate depends on the kernel and its adaptivity transformation, in order to achieve convergence, both must be kept independent of the fill distance. Stationary refinement would be perhaps the simplest example of a fill distance dependent adaptivity transformation, where the kernel would be taken to be

$$\Phi_h(x,y) := \hat{\Phi}\left(h^{-1}x, h^{-1}y\right) \text{ for all } (x,y) \in \Omega \times \Omega. \tag{2.10}$$

The main purpose of stationary refinement is that it allows for a fixed cost of computation per discretization point, and furthermore typically results in a more stable interpolation problem. In the context of kernel-based interpolation, this is a well-studied problem, c.f. [24, Section 15.4], for which convergence results are limited to certain globally-supported kernels. In the crucial case of compactly-supported kernels there are no known convergence results for stationary refinement. Since adaptive error estimates for local stationary refinement would imply convergence for global stationary refinement, such estimates seem unlikely, and are not attempted here.

### 2.3.3 Stability of Interpolation Using Adaptive Kernels

Because the interpolation matrices coincide their stability properties do as well, and therefore we can study the stability of $A_{\Phi,X}$ via $A_{\hat{\Phi},\hat{X}}$. As discussed by Wendland [75, Chapter 12], the condition number of the interpolation matrix is given by the ratio of its maximum eigenvalue to its minimum eigenvalue. The maximum eigenvalue is well-behaved, growing like $h\left(\hat{X},\hat{\Omega}\right)^{-n}$, for a scale of point distributions $\hat{X}$ with a uniformly bounded mesh ratio (as the fill distance of $\hat{X}$ decreases, the ratio between the fill and separation distances remains bounded from above and below by certain positive constants). The minimum eigenvalue on the other hand is bounded from below in terms of the *separation distance*,

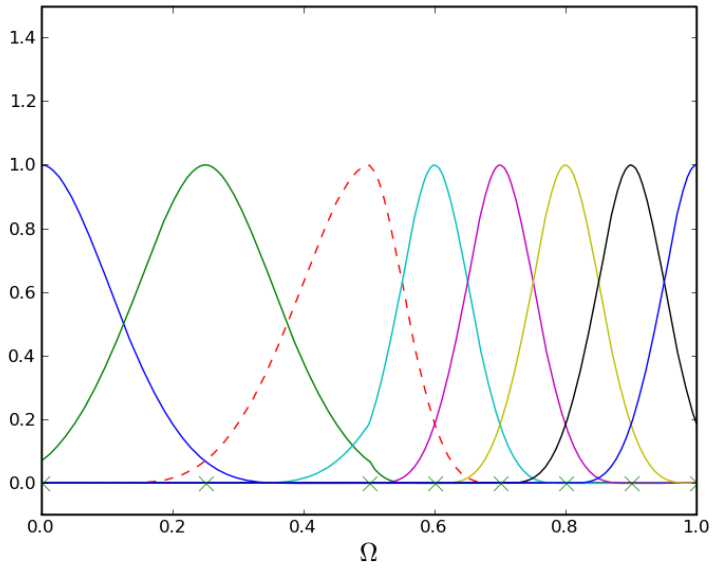
$$q := q\left(X\right) := \frac{1}{2}\min_{x,y\in X, x\neq y}\left\|x-y\right\|.$$

In the case that $\hat{\Phi}$ is the compactly-supported $(n,k)$-Wendland function, it was shown that $\lambda_{\min}\left(A_{\hat{\Phi},\hat{X}}\right) \geq C\left(n,k\right)q\left(\hat{X}\right)^{2k+1}$. Thus, if $q\left(\hat{X}\right) \geq q\left(X\right)$ it follows that interpolation using the adaptively-scaled kernel is more stable than using a nonadaptively-scaled kernel. Examples of this are provided in Section 2.4.

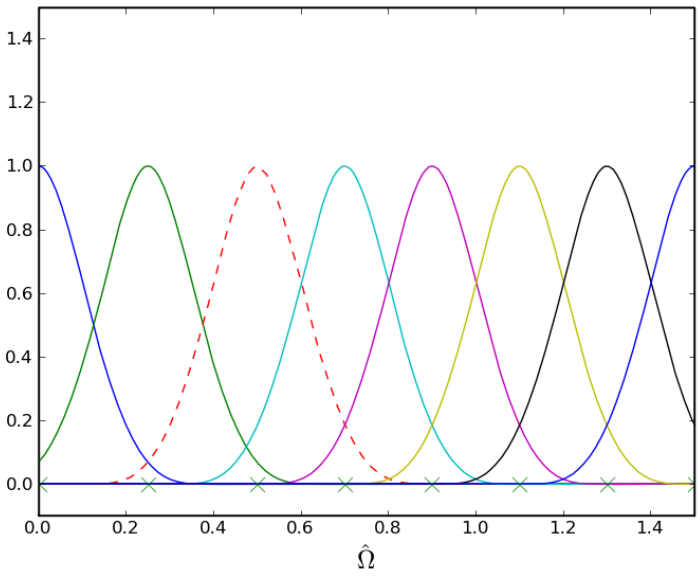
## 2.4 Computational Examples

In the first example we consider the domain $\Omega = (0,1)$, with $\hat{\Omega} = \left(0,\frac{3}{2}\right)$. These domains are each partitioned into two subdomains. We take $\Omega_1 = \left(0,\frac{1}{2}\right)$ and $\Omega_2 = \left(\frac{1}{2},1\right)$, while $\hat{\Omega}_1 = \left(0,\frac{1}{2}\right)$ and $\hat{\Omega}_2 = \left(\frac{1}{2},\frac{3}{2}\right)$. Over $\hat{\Omega}_1$ the adaptivity transformation is just the identity, while for $x \in \hat{\Omega}_2$ it is defined by

$$T : x \mapsto \frac{1}{2}\left(x+\frac{1}{2}\right) \tag{2.11}$$

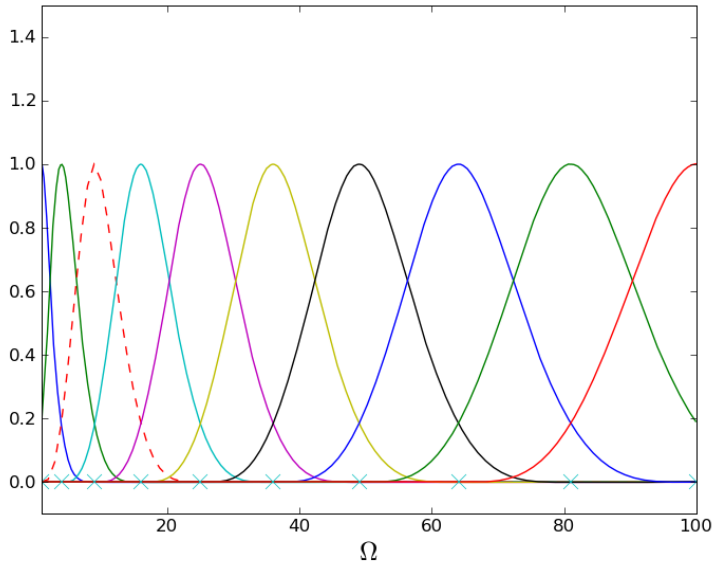



(a) A trial space over the domain $\Omega = (0, 1)$ which is defined in terms of a kernel which has been adaptively-scaled via the transformation (2.11). The dashed trial function is scaled non-uniformly within its support. The size of its support in $\Omega_1 = \left(0, \frac{1}{2}\right)$ is twice that of its support in $\Omega_2 = \left(\frac{1}{2}, 1\right)$.

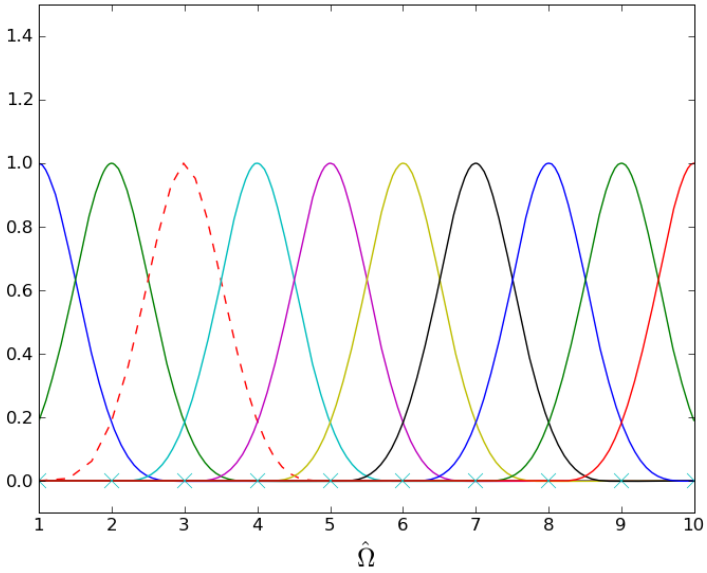



(b) The corresponding trial space over the domain $\hat{\Omega} = \left(0, \frac{3}{2}\right)$ which is defined in terms of a translation-invariant kernel. The separation distance is $\frac{1}{8}$, while the condition number is 2.024.

Figure 2.4: Illustration of well-posed interpolation using adaptively-scaled kernels.

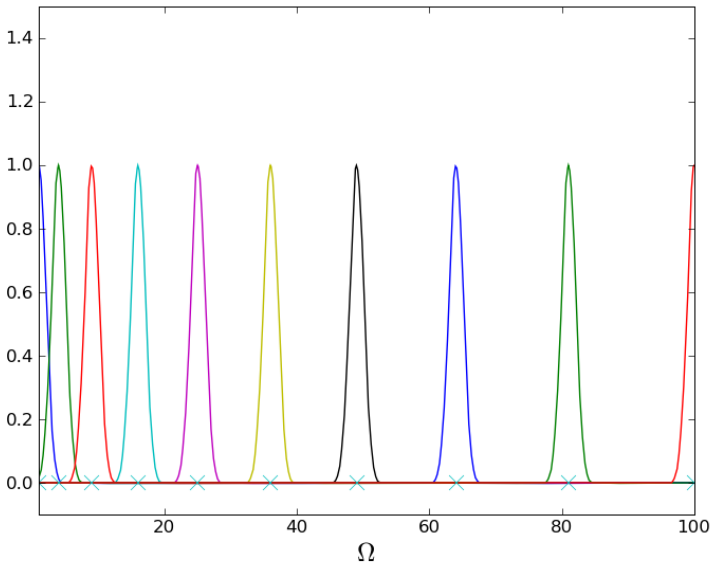



(a) A trial space over the domain $\Omega = (1, 100)$ which is defined in terms of a kernel which has been adaptively-scaled via the transformation (2.12).

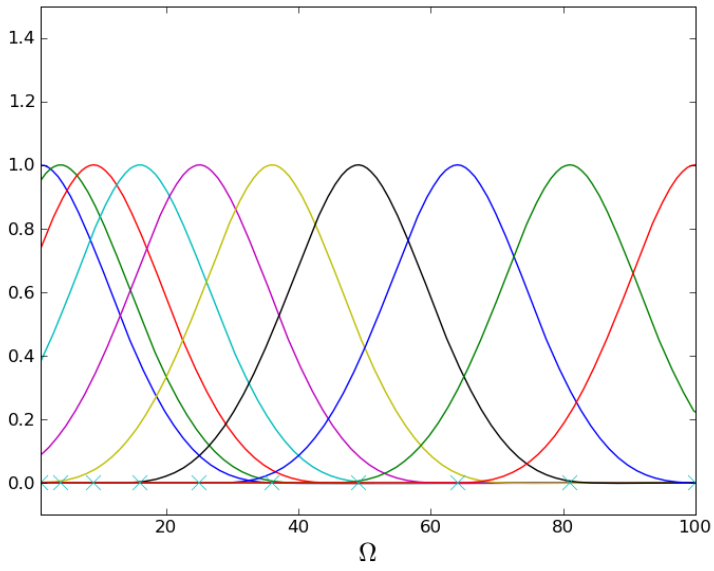



(b) The corresponding trial space over the domain $\hat{\Omega} = (1, 10)$ which is defined in terms of a translation-invariant kernel. The separation distance is $\frac{1}{2}$, while the condition number is 2.124.

Figure 2.5: Illustration of well-posed interpolation using adaptively-scaled kernels.




(a) "Too small" of a kernel size has been chosen for all but the leftmost trial function. The condition number of the associated interpolation matrix is 1.032.




(b) "Too large" of a kernel size has been for all but the rightmost trial function. The condition number of the associated interpolation matrix is 263.278.

Figure 2.6: Interpolation using a nonadaptively-scaled translation-invariant kernel (1.5). A fixed kernel size must be chosen which is either "too small" (2.6a) or "too large" (2.6b).

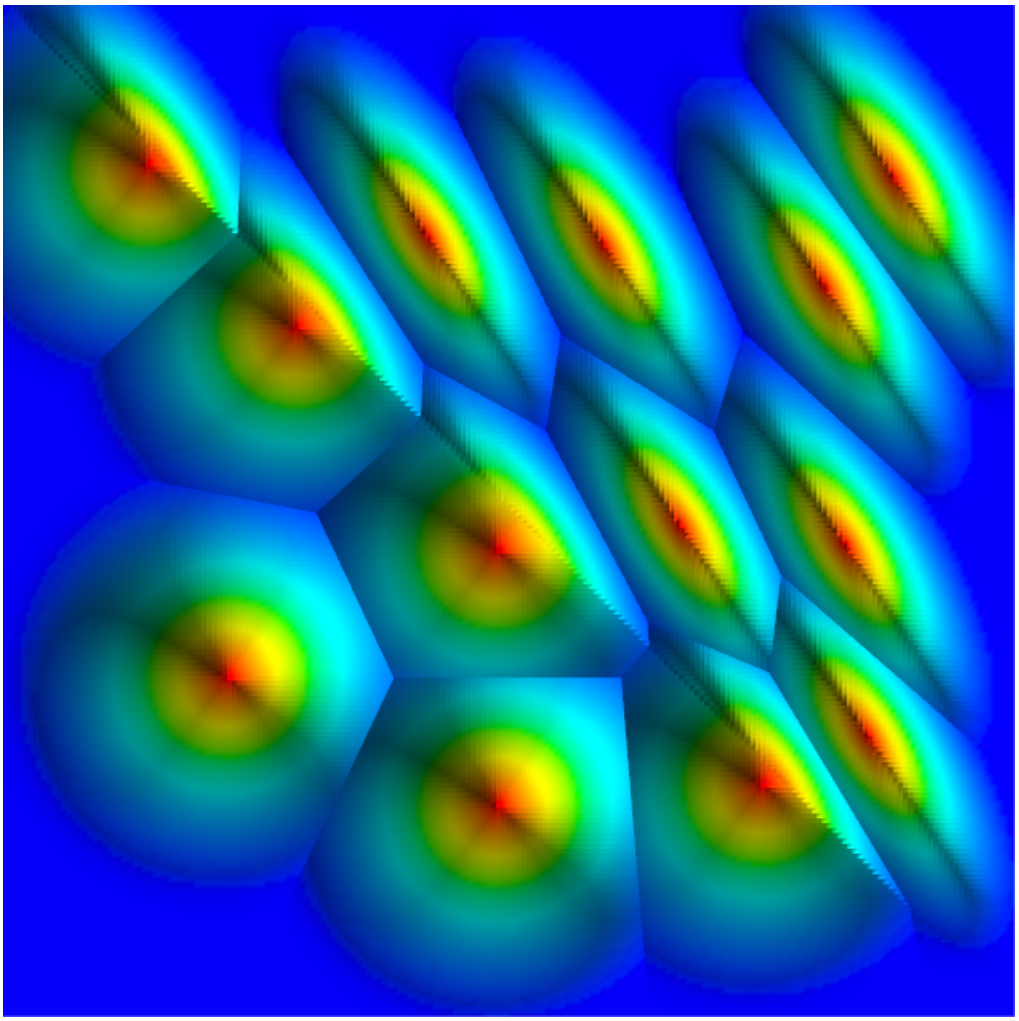
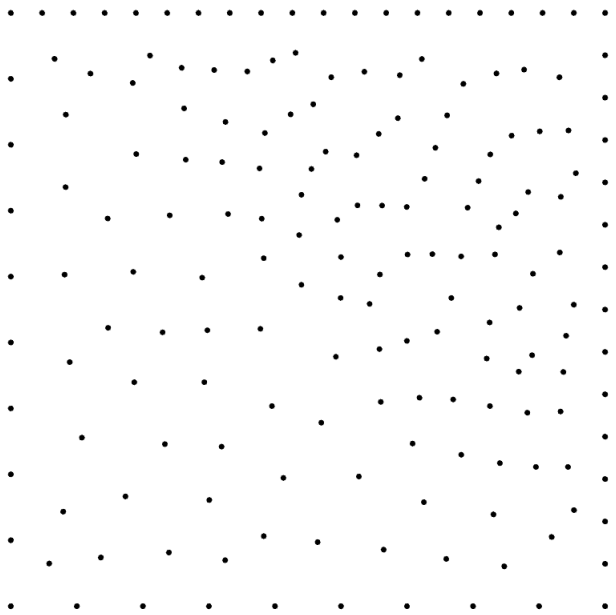
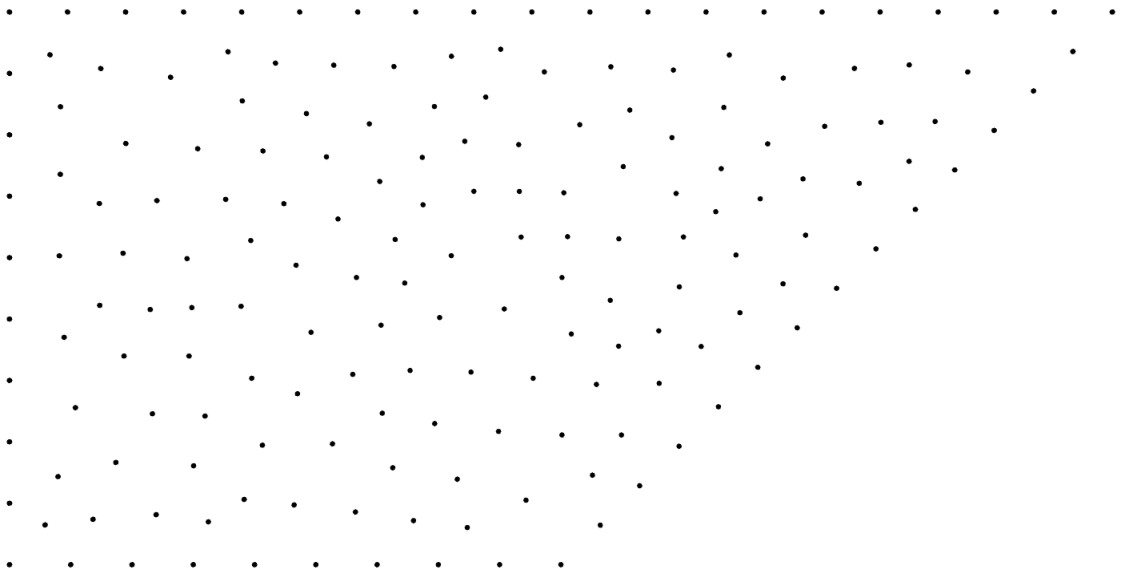


Figure 2.7: Adaptively-scaled trial functions in two dimensions. The cell structure which is visible is merely due to the overlap of the trial functions' supports.



(a) An irregularly distributed set of points $X$ in a two-dimensional domain $\Omega$.



(b) The corresponding regularly distributed set of points $\hat{X}$ in the corresponding domain $\hat{\Omega}$.

Figure 2.8: An illustration of an adaptivity transformation in two dimensions.

Figure 2.4a illustrates $X$ and $\Omega$, along with examples of the trial functions. Trial functions whose supports are contained completely within either $\Omega_1$ or $\Omega_2$ are just shifted and uniformly-scaled Wendland functions [75, Section 9.4]. However, trial functions whose support intersects both sub-domains are scaled differently within each. This type of scaling differs from the more traditional approach to adaptive scaling, described in Section 2.1 and illustrated in Figure 2.1, where each trial function would be scaled uniformly throughout its support. We also observe the improved stability using adaptively-scaled kernels by comparing the interpolation matrix condition numbers corresponding to Figures 2.4 and 2.2b.

In the second example we consider the domain $\Omega = (1, 100)$, with $\hat{\Omega} = (1, 10)$. The adaptivity transformation is the mapping

$$T : x \mapsto x^2 \tag{2.12}$$

Figure 2.5a illustrates $X$ and $\Omega$, along with examples of the trial functions. We observe the improved stability using adaptively-scaled kernels by comparing the interpolation matrix condition numbers corresponding to Figures 2.5 and 2.6b.

In the third example we consider the domain $\Omega = \Omega_1 \cup \Omega_2$, with $\hat{\Omega} = \hat{\Omega}_1 \cup \hat{\Omega}_2$. Let $\Omega_1, \hat{\Omega}_1$ be the triangle defined by vertices $((0,0), (1,0), (0,1))$, $\Omega_2$ be the triangle defined by vertices $((1,0), (1,1), (0,1))$, and $\hat{\Omega}_2$ be the triangle defined by vertices $((1,0), (2,1), (0,1))$. The adaptivity transformation $T$ is the piecewise affine transformation such that $T$ maps $\hat{\Omega}_i$ to $\Omega_i$ by an affine mapping for $i = 1, 2$. Its inverse is used computationally, and can be implemented as follows. Given a point $x \in \Omega_i$, its barycentric coordinates are computed, i.e., $\lambda_1, \lambda_2, \lambda_3$, such that $x = \sum_{i=1}^{3} \lambda_i v_i$ where $v_i$ are the vertices of the triangle $\Omega_i$. The corresponding point $\hat{x}$ is then given by $\sum_{i=1}^{3} \lambda_i \hat{v}_i$ where $\hat{v}_i$ are the vertices of the triangle $\hat{\Omega}_i$. A trial space defined using this adaptivity transformation is illustrated in Figure 2.7. A point distribution for which for which this adaptivity transformation increases the separation distance and thus improves conditioning is illustrated in Figure 2.8.

## 2.5 Conclusions

The technique suggested here is the first well-posed, general technique for kernel-based interpolation with adaptively-scaled trial functions. Using an appropriately scaled kernel, which counteracts local variations in the density of the underlying point distribution, can increase the separation distance and thus improve stability.

We have provided examples in one and two dimensions which use a manually-defined adaptivity transformation. Future research should provide techniques for constructing arbitrary-order adaptivity transformations in higher dimensions automatically in terms of a given meshless point distribution. This would preferably be done using common existing spatial data structures such as quadtrees or octrees. Although it would probably employ a mesh in some form, the meshing requirements would be minimal as the mesh would not need to conform to the domain or be as refined as the underlying meshless point distribution.

In the context of moving least squares approximation, kernels are used as a weighting function. In the past, these have usually been translation-invariant functions, which do not lead to adaptivity. However, since moving least squares approximation is capable of stationary refinement, future work will investigate adaptive error estimates for moving least squares approximation. Furthermore, an important property of the adaptively-scaled kernel introduced here is that it is symmetric, which is essential in obeying conservation laws for mimetic numerical methods.

As in the case of global stationary refinement one might expect some increase in accuracy using local stationary refinement prior to reaching *saturation* (in the sense of the Maz'ya and Schmidt theory of approximate approximation [46], see also [24, Section 17.3]). The special case of global stationary refinement appears to limit the possibility of establishing adaptive error estimates for local stationary refinement.

# Chapter 3: A Sampling Inequality for Fractional Order Sobolev Semi-Norms Using Arbitrary Order Data

## 3.1 Introduction

Over the past few years, increasingly general bounds of Sobolev semi-norms, in terms of discrete samples, have appeared. Such bounds are often called *sampling inequalities*. A rather general sampling inequality was established by Arcangéli et al. [2], and is stated as Theorem 3.1, with notation given in Section 3.1.1.

**Theorem 3.1.** [2, Theorem 4.1] Let $\Omega$ be a Lipschitz domain in $\mathbb{R}^n$, so that the domain $\Omega$ satisfies the cone property [2, Page 185] with radius $\rho > 0$ and angle $\theta \in (0, \pi/2]$. Furthermore, let $p, q, \varkappa \in [1, \infty]$ and let $r$ be a real number such that $r \geq n$, if $p = 1$, $r > n/p$ if $1 < p < \infty$, or $r \in \mathbb{N}^*$, if $p = \infty$. Let $l_0 = r - n\,(1/p - 1/q)_+$ and $\gamma = \max\{p, q, \varkappa\}$. Then, there exist two positive constants $\mathfrak{d}_r$ (*dependent on* $\theta, \rho, n$ and $r$) and $C$ (*dependent on* $\Omega, n, r, p, q$ and $\varkappa$) satisfying the following property: for any set $A \subset \overline{\Omega}$ (or $A \subset \Omega$ if $p = 1$ and $r = n$) such that $d = \delta\left(A, \overline{\Omega}\right) \leq \mathfrak{d}_r$ (c.f. (3.2)) for any $u \in W^{r,p}(\Omega)$ and for any real number $l$ satisfying $l = 0, \ldots, l_{max}$, we have

$$|u|_{l,q,\Omega} \leq C\left(d^{r-l-n(1/p-1/q)_+}\,|u|_{r,p,\Omega} + d^{n/\gamma-l}\,\|\,|u|_b\|_{\varkappa}\right). \tag{3.1}$$

where $l_{\max} := \lceil l_0 \rceil - 1$, unless the following additional conditions hold, in which case $l_{\max} := l_0$: $r \in \mathbb{N}^*$ and either (i) $p < q < \infty$ and $l_0 \in \mathbb{N}$, (ii) $(p, q) = (1, \infty)$, or (iii) $p \geq q$.

This sampling inequality generalizes those of Madych [44] and Wendland and Rieger [73], by greatly extending the range of parameters $r, p, l$, and $\varkappa$. While Theorem 3.1 applies to

functions with finite smoothness, an analogous bound for functions with infinite smoothness has been provided by Rieger and Zwicknagl [59] which achieves exponential factors.

Arcangéli, et al. [2] used Theorem 3.1 to derive error bounds for interpolating and smoothing $(m, s)$-splines, an application which we do not consider. Instead we are interested in another major application of these Sobolev estimates: Schaback's framework for unsymmetric meshless methods for operator equations [63], see also the earlier version [64]. A sampling inequality is necessary for unsymmetric meshless methods, such as Schaback's modification of Kansa's method [63, 64], which involve an overdetermined system of equations in general. In an attempt to improve the order of convergence obtained using Schaback's framework, we extend the bound of Arcangéli, et al. in two ways.

Our first extension is to loosen the restriction $l \in \mathbb{N}$ to allow for fractional order Sobolev norms on the left hand side of the sampling inequality. In the context of Schaback's framework, this will result in more optimal convergence results in terms of both the test and trial discretization parameters. Otherwise, the test discretization would require a higher rate of refinement.

Our second extension is to incorporate discrete samples of arbitrary order derivatives into the bound. The reason for this is that (3.1) has a factor $d^{-l}$ in its second term which is insufficient for achieving a *uniformly stable test discretization* for higher order Sobolev norms in Schaback's framework. With this modification to incorporate samples of higher order derivatives we will be able to come closer to achieving such a test discretization, resulting in higher order convergence results. This introduces a new parameter $\mu$, for which previous sampling inequalities coincide with the choice $\mu = 0$.

### 3.1.1 Notation

We largely use the notation of Arcangéli, et al. [2, Section 2]. Here we restate a portion of their notation. For all $r \in [0, \infty)$ and $p \in [1, \infty]$, the Sobolev norm is denoted by $\|\cdot\|_{r,p,\Omega}$, while the Sobolev semi-norm is denoted by $|\cdot|_{r,p,\Omega}$. The set $\mathbb{N}^* = \{1, 2, 3, \ldots\}$, while $\mathbb{N} = \{0, 1, 2, \ldots\}$. As in [2, Section 2], if $N \in \mathbb{N}^*$, $\varkappa \in [1, \infty]$, $b = (b_1, \ldots, b_N) \in (\mathbb{R}^n)^N$ and

a function $v$ has well-defined values on each point $b_j$, then the discrete norm of $v$ over $b$ is given by

$$\|v|_b\|_\varkappa = \begin{cases} \left(\sum_{j=1}^N |v(b_j)|^\varkappa\right)^{1/\varkappa}, & \text{if} \varkappa \in [1,\infty) \\ \max_{1\le j\le N} |v(b_j)| & \text{if} \varkappa = \infty \end{cases}$$

The floor of a real number $r$ is denoted by $\lfloor r \rfloor$ and is the largest integer $k$ such that $k \le r$, while its ceiling is denoted by $\lceil r \rceil$ and is the smallest integer $k$ such that $k \ge r$. The space of polynomials over $\mathbb{R}^n$ with degree less than or equal to $k$ is denoted by $P_k$.

As in [2, Section 4], we assume throughout this chapter that $\Omega$ is a bounded domain in $\mathbb{R}^n$ with a Lipschitz-continuous boundary, so that the domain $\Omega$ satisfies the cone property [2, Page 185] with radius $\rho > 0$ and angle $\theta \in (0, \pi/2]$. For a given finite subset $A$ of $\overline{\Omega}$, the *fill distance* is defined as

$$\delta\left(A, \overline{\Omega}\right) = \sup_{x\in\Omega} \min_{a\in A} |x - a|. \tag{3.2}$$

In order to follow the notation of Arcangéli, et al. [2], in this chapter we will not use the notation $h(X, \Omega)$.

We make the following additions to their notation. Let $H^r(\Omega) := W^{r,2}(\Omega)$ and

$$\tilde{W}^{r,q}(\Omega) := \left\{ v \in W^{r,q}(\Omega) \,:\, \int_\Omega v = 0 \right\}.$$

Given a function $v \in W^{1,q}(\Omega)$, the vector-valued function consisting of its partial derivatives is denoted by $Dv$. The surface area of the $n$-dimensional ball is denoted by $\left|S^{n-1}\right|$. In Section 3.2, a generic constant $C$ appears in many proofs, whose particular value may change, but with the parameters on which it depends either indicated in parentheses or stated explicitly in the exposition. We will often substitute dependencies with others, possibly taking the maximum or minimum value, as required by its application, of the constant over a finite range of values. In Section 3.3, only the dependence of constants on

the discretization parameters $r$ and $s$ is explicitly stated since we regard the spaces and mappings in that section as fixed.

## 3.2 Extension of the Sobolev Bound

### 3.2.1 Fractional Order Sobolev Spaces

This section concerns fractional order Sobolev norms and the results of this section will be used to generalize [2, Proposition 3.4] to Proposition 5. Lemmas 2 and 3 each require an extension operator which satisfies (3.4) for zero-average functions over a ball. This property is not provided by standard extension operators since they involve a domain-dependent constant and the full Sobolev norm in the bound, rather than a domain-independent constant and the Sobolev semi-norm.

**Lemma 1.** If $q \in [1, \infty]$, $r > 0$, and $x_0 \in \mathbb{R}^n$ then there exists a linear, continuous operator

$$E : \tilde{W}^{1,q}\left(B\left(x_0, r\right)\right) \to W^{1,q}\left(\mathbb{R}^n\right)$$

such that for all $v \in \tilde{W}^{1,q}\left(B\left(x_0, r\right)\right)$

$$Ev = v \text{ a.e. in } B\left(x_0, r\right), \tag{3.3}$$

and

$$\left|Ev\right|_{1,q,\mathbb{R}^n} \leq C\left(n, q\right)\left|v\right|_{1,q,B(x_0,r)}. \tag{3.4}$$

If in addition $v \in C^1\left(\overline{B\left(x_0, r\right)}\right)$ then $Ev \in C^1\left(\mathbb{R}^n\right)$.

*Proof.* Let $v \in \tilde{W}^{1,q}\left(B\left(x_0, r\right)\right)$ and $\hat{v} := v \circ F$ where $F : \hat{x} \to r\hat{x} + x_0$. From a change of variables it follows that $\hat{v} \in W^{1,q}\left(B\left(0, 1\right)\right)$ with semi-norm

$$\left|\hat{v}\right|_{1,q,B(0,1)} = r^{1-n/q}\left|v\right|_{1,q,B(x_0,r)}.$$

From [22, Section 5.4, Theorem 1], there exists a linear, continuous extension operator

$$\hat{E} : W^{1,q}\left(B\left(0,1\right)\right) \rightarrow W^{1,q}\left(\mathbb{R}^n\right)$$

such that for each $v \in W^{1,q}\left(B\left(0,1\right)\right)$

$$\hat{E}\hat{v} = \hat{v} \text{ a.e. in } B\left(0,1\right),$$

and

$$\left\|\hat{E}\hat{v}\right\|_{1,q,\mathbb{R}^n} \leq C\left(n,q\right)\left\|\hat{v}\right\|_{1,q,B(0,1)},$$

where the dependence on $n$ is through $B\left(0,1\right)$. That

$$\left\|\hat{v}\right\|_{1,q,B(0,1)} \quad \leq \quad C\left(n,q\right)\left|\hat{v}\right|_{1,q,B(0,1)},$$

follows from specializing a Poincaré inequality given in [22, Section 5.8, Theorem 2] to the unit ball and that

$$\int_{B(0,1)} \hat{v} = r^{-n}\int_{B(x_0,r)} v = 0, \tag{3.5}$$

it follow Let

$$Ev := \left(\hat{E}\hat{v}\right) \circ F^{-1} \tag{3.6}$$

so that the result (3.3) holds. From another change of variables, it follows that

$$\left|Ev\right|_{1,q,\mathbb{R}^n} = r^{n/q-1}\left|\hat{E}\hat{v}\right|_{1,q,\mathbb{R}^n}.$$

The result (3.4) follows by combining the preceding relations. Finally, from the proof of [22, Section 5.4, Theorem 1] it follows that if $\hat{v} \in C^1\left(\overline{B\left(0,1\right)}\right)$, then $\hat{E}\hat{v} \in C^1\left(\mathbb{R}^n\right)$, so

that if $v \in C^1\left(\overline{B\left(x_0, r\right)}\right)$ then $Ev \in C^1\left(\mathbb{R}^n\right)$. $\square$

Based on this extension, we obtain Lemma 2, which is similar to a result used by Bourgain et al. [6, Eq. 2], but uses a domain-independent constant and semi-norm in the bound.

**Lemma 2.** If $q \in [1, \infty)$, $h \in \mathbb{R}^n$, and $v \in \tilde{W}^{1,q}\left(B\left(x_0, r\right)\right)$ then

$$\left(\int_{B(x_0,r)} \left|Ev\left(x+h\right) - Ev\left(x\right)\right|^q dx\right)^{1/q} \leq C\left(n, q\right)|h|\,|v|_{1,q,B(x_0,r)}$$

*Proof.* First suppose that $v$ is in the subset

$$C^1\left(\overline{B\left(x_0, r\right)}\right) \cap \tilde{W}^{1,q}\left(B\left(x_0, r\right)\right) \tag{3.7}$$

which is dense in $\tilde{W}^{1,q}(B(x_0,r))$. From Lemma 1 it follows that

$$\begin{aligned}
\int_{B(x_0,r)} |Ev(x+h)-Ev(x)|^q\,dx &\leq \int_{\mathbb{R}^n} |Ev(x+h)-Ev(x)|^q\,dx \\
&= \int_{\mathbb{R}^n} \left|\int_0^1 \frac{d}{dt}Ev(x+th)\,dt\right|^q dx \\
&\leq \int_{\mathbb{R}^n}\int_0^1 \left|\frac{d}{dt}Ev(x+th)\right|^q dtdx \\
&= \int_{\mathbb{R}^n}\int_0^1 |DEv(x+th)\cdot h|^q\,dtdx \\
&\leq |h|^q \int_{\mathbb{R}^n}\int_0^1 |DEv(x+th)|^q\,dtdx \\
&= |h|^q \int_0^1\int_{\mathbb{R}^n} |DEv(x+th)|^q\,dxdt \\
&\leq |h|^q \int_0^1 |Ev(\cdot+th)|^q_{1,q,\mathbb{R}^n}\,dt \\
&= |h|^q\,|Ev|^q_{1,q,\mathbb{R}^n} \\
&\leq C(n,q)\,|h|^q\,|v|^q_{1,q,B(x_0,r)}
\end{aligned}$$

From this, the result follows for all functions in $\tilde{W}^{1,q}(B(x_0,r))$ via a standard density argument. □

**Lemma 3.** If $x, x+h \in B(x_0,r)$, and $v \in \tilde{W}^{1,\infty}(B(x_0,r))$ then

$$|v(x+h)-v(x)| \leq C(n)\,|h|\,|v|_{1,\infty,B(x_0,r)}$$

*Proof.* In the proof of [22, Section 5.8.2, Theorem 4] it is shown that $v$ is a Lipschitz function with constant $|Ev|_{1,\infty,\mathbb{R}^n}$, where the extension operator constructed in [22, Section 5.4, Theorem 1] is used. However, the extension operator from Lemma 1 could be substituted

so that the result then follows by (3.4). □

In the bound provided by Lemma 4 the factor $r^{1-\epsilon}$ will be the key to generalizing sampling inequalities to optimally bound fractional order semi-norms.

**Proposition 4.** *If* $q \in [1, \infty]$, $\epsilon \in (0, 1)$, *and* $v \in W^{1,q}(B(x_0, r))$ *then*

$$|v|_{\epsilon,q,B(x_0,r)} \leq C(n, q)(1-\epsilon)^{-1/q} r^{1-\epsilon} |v|_{1,q,B(x_0,r)}. \tag{3.8}$$

*Proof.* Since the semi-norms that appear in (3.8) are invariant with respect to a shift in value of $v$ by a constant, it suffices to only consider $v \in \tilde{W}^{1,q}(B(x_0, r))$.

Case $q \in [1, \infty)$: Let $y \in B(x_0, r)$ and

$$B(x_0, r) - y := \{x - y : x \in B(x_0, r)\},$$

so that $B(x_0, r) - y \subseteq B(0, 2r)$.

$$
\begin{aligned}
|v|^q_{\epsilon,q,B(x_0,r)} &= \int_{B(x_0,r)}\int_{B(x_0,r)} \frac{|v(x)-v(y)|^q}{|x-y|^{n+\epsilon q}}dxdy \\
&= \int_{B(x_0,r)}\int_{B(x_0,r)-y} \frac{|v(y+h)-v(y)|^q}{|h|^{n+\epsilon q}}dhdy \\
&\leq \int_{B(x_0,r)}\int_{B(0,2r)} \frac{|Ev(y+h)-Ev(y)|^q}{|h|^{n+\epsilon q}}dhdy \\
&= \int_{B(0,2r)} \frac{\int_{B(x_0,r)}|Ev(y+h)-Ev(y)|^q\,dy}{|h|^{n+\epsilon q}}dh \\
&\leq C(n,q)\left(\int_{B(0,2r)}\frac{|h|^q}{|h|^{n+\epsilon q}}dh\right)|v|^q_{1,q,B(x_0,r)} \\
&= C(n,q)\left|S^{n-1}\right|\left(\int_0^{2r}\frac{\rho^q}{\rho^{n+\epsilon q}}\rho^{n-1}d\rho\right)|v|^q_{1,q,B(x_0,r)} \\
&\leq \frac{C(n,q)}{(1-\epsilon)q}\left|S^{n-1}\right|(2r)^{(1-\epsilon)q}|v|^q_{1,q,B(x_0,r)} \\
&\leq \frac{C(n,q)}{(1-\epsilon)q}\left|S^{n-1}\right|2^q r^{(1-\epsilon)q}|v|^q_{1,q,B(x_0,r)}.
\end{aligned}
$$

Case $q=\infty$:

$$
\begin{aligned}
|v|_{\epsilon,q,B(x_0,r)} &= \operatorname*{ess\,sup}_{x,y\in B(x_0,r),x\neq y}\frac{|v(x)-v(y)|}{|x-y|^\epsilon} \\
&\leq C(n,q)(2r)^{1-\epsilon}|v|_{1,q,B(x_0,r)} \\
&\leq 2C(n,q)\,r^{1-\epsilon}|v|_{1,q,B(x_0,r)}.
\end{aligned}
$$

□

**Remark 3.1.** The explicit constant $(1-\epsilon)^{-1/q}$, which blows up as $\epsilon$ increases towards one, is a manifestation of the "defect" of intrinsic fractional order Sobolev semi-norms studied

by Bourgain et al. [6].

**Corollary 1.** If $q \in [1,\infty]$, $l \in [0,\infty]$, and $v \in W^{\lceil l \rceil, q}\left(B\left(x_0, r\right)\right)$ then

$$|v|_{l,q,B(x_0,r)} \leq C\left(n, q, \lfloor l \rfloor\right) K\left(\lceil l \rceil - l, q\right) r^{\lceil l \rceil - l} \left|v\right|_{\lceil l \rceil, q, B(x_0,r)}, \tag{3.9}$$

where

$$K\left(\lceil l \rceil - l, q\right) := \begin{cases} 1 & \text{for } l \in \mathbb{N} \text{ or } q = \infty \\ \left(\lceil l \rceil - l\right)^{-1/q} & \text{for } l \notin \mathbb{N} \text{ and } q < \infty \end{cases}. \tag{3.10}$$

### 3.2.2 An Auxiliary Result

The following result applies Corollary 1 to generalize [2, Proposition 3.4].

**Proposition 5.** *Let $p, q, \varkappa \in [1,\infty]$ such that $p \leq q$. Let $r$ be a real number such that $r > n/p$, if $p > 1$, or $r \geq n$, if $p = 1$. Finally, let $k = \lceil r \rceil - 1$, $\mathfrak{K} = \dim P_k$, and $l_0 = r - n/p + n/q$. Then, there exists a constant $R > 1$ (dependent on $n$ and $r$) and, for any $M' \geq 1$, there exists two constants $C$ (dependent on $M', n, r, p, q$, and $\varkappa$) and $K \geq 1$* (explicitly dependent on $\lceil l \rceil - l$ and $q$, cf. (3.10)), *satisfying the following property: for any $d > 0$ and any $t \in \mathbb{R}^n$, the open ball $B\left(t, Rd\right)$ contains $\mathfrak{K}$ closed balls $\mathcal{B}_1, \ldots \mathcal{B}_{\mathfrak{K}}$ of radius $d$ such that, for any $v \in W^{r,p}\left(\overline{B}\left(t, M'Rd\right)\right)$, for any $b \in \Pi_{i=1}^{\mathfrak{K}} \mathcal{B}_i$ and $l \in [0, l_{\max}]$,*

$$|v|_{l,q,\overline{B}(t,M'Rd)} \leq C \cdot K\left(d^{r-l-n/p+n/q} \left|v\right|_{r,p,\overline{B}(t,M'Rd)} + d^{n/q-l} \left\| v|_b \right\|_{\varkappa}\right), \tag{3.11}$$

*where we have let $l_{\max} := \lceil l_0 \rceil - 1$, or $l_{\max} := l_0$ if the following additional conditions hold: $r \in \mathbb{N}^*$ and either (i) $p < q < \infty$ and $l_0 \in \mathbb{N}$, (ii) $(p,q) = (1,\infty)$, or (iii) $1 \leq p = q \leq \infty$.*

*Proof.* The case that $l \in \mathbb{N}$ is established by [2, Proposition 3.4]. Suppose that $l \notin \mathbb{N}$. The hypotheses imply that $l_{\max} \in \mathbb{N}$, so that $\lceil l \rceil \leq l_{\max}$, and thus the result follows by combining (3.11) for $l = \lceil l \rceil$ with Corollary 1, using the fact that $\left(M'R\right)^{\lceil l \rceil - l} \leq M'R$, and

substituting the dependence on $\lfloor l \rfloor$ and $R$ with $n, r, p$ and $q$. $\square$

### 3.2.3 Sobolev Bounds

For $p \leq q$, the following result generalizes [2, Theorem 4.1] to bound fractional order Sobolev semi-norms. No generalization to bound fractional order Sobolev semi-norms is made for $p > q$, since we have not obtained the relation [2, Eq. 2.1] for the case that $l$ is fractional.

**Theorem 3.2.** Let $p, q, \varkappa \in [1, \infty]$ and let $r$ be a real number and $\mu$ a nonnegative integer such that $r - \mu \geq n$, if $p = 1$, $r - \mu > n/p$ if $1 < p < \infty$, or $r - \mu \in \mathbb{N}^*$ if $p = \infty$. Let $l_0 = r - \mu - n\,(1/p - 1/q)_+$ and $\gamma = \max\{p, q, \varkappa\}$. Then, there exist three positive constants $\mathfrak{d}_r$ (dependent on $\theta, \rho, n, r$ and $\mu$), $C$ (dependent on $\Omega, n, r, p, q$, and $\varkappa$), and $K \geq 1$ (explicitly dependent on $\lceil l \rceil - l$ and $q$, cf. (3.10)), satisfying the following property: for any set $A \subset \overline{\Omega}$ (or $A \subset \Omega$ if $p = 1$ and $r - \mu = n$) such that $d = \delta\left(A, \overline{\Omega}\right) \leq \mathfrak{d}_r$, for any $u \in W^{r,p}(\Omega)$ and, if $p \leq q$ then for any real number $l \in [0, l_{\max}]$, otherwise if $p > q$ then for any integer $l = 0, \ldots, l_{\max}$,

$$|u|_{l,q,\Omega} \leq C \cdot K \left( d^{r-l-n(1/p-1/q)_+} |u|_{r,p,\Omega} + d^{n/\gamma+\mu-l} \left\| \prod_{|\alpha|=\mu} \partial^\alpha u|_A \right\|_\varkappa \right), \tag{3.12}$$

where we have let $l_{\max} := \lceil l_0 \rceil - 1$, or $l_{\max} := l_0$ if the following additional conditions hold: $r \in \mathbb{N}^*$ and either (i) $p < q < \infty$ and $l_0 \in \mathbb{N}$, (ii) $(p, q) = (1, \infty)$, or (iii) $p \geq q$.

*Proof.* The case that $\mu = 0$ and $l \in \mathbb{N}$ is [2, Theorem 4.1]. The proof of this theorem for $\mu = 0$ and $l \notin \mathbb{N}$ can be obtained by reusing the proof of [2, Theorem 4.1], but applying Proposition 5 instead of [2, Proposition 3.4], which allows for $l$ to be of fractional order for $p \leq q$, and introduces the constant $K$. We now consider the case $\mu > 0$. Let $\alpha$ be a multi-index such that $|\alpha| = \mu$ and therefore $\partial^\alpha u \in W^{r-\mu,p}(\Omega)$. It follows from the case

that $\mu = 0$ that in the situation required by the present hypotheses

$$|\partial^\alpha u|_{l-\mu,q,\Omega} \leq C \cdot K \left( d^{r-l-n(1/p-1/q)_+} |\partial^\alpha u|_{r-\mu,p,\Omega} + d^{n/\gamma+\mu-l} \| \partial^\alpha u|_A \|_\varkappa \right).$$

We have for all $\alpha$ satisfying $|\alpha| = \mu$ that

$$|\partial^\alpha u|_{r-\mu,p,\Omega} \leq |u|_{r,p,\Omega}$$

and

$$\| \partial^\alpha u|_A \|_\varkappa \leq \left\| \prod_{|\beta|=\mu} \partial^\beta u \Big|_A \right\|_\varkappa .$$

The result follows immediately for $q = \infty$. Otherwise, if $1 \leq q < \infty$, using that

$$|u|^q_{l,q,\Omega} \leq \sum_{|\alpha|=\mu} |\partial^\alpha u|^q_{l-\mu,q,\Omega}$$

the results follows with an additional factor $(\#\{\alpha : |\alpha| = \mu\})^{1/q}$ in the constant $C$, whose dependence on $\mu$ can be substituted with $n, r$ and $p$. □

**Corollary 2.** Given the situation of Theorem 3.2 with a constant $\mathfrak{d}_r$ now dependent on $\theta, \rho, n, r, p$ and $q$, and the additional assumption that $r - l \in \mathbb{N}$, then we have

$$\|u\|_{l,q,\Omega} \leq C \cdot K \left( d^{r-l-n/p+n/q} \|u\|_{r,p,\Omega} + d^{n/\gamma+\mu-l} \left\| \prod_{|\alpha|\leq\mu} \partial^\alpha u|_A \right\|_\varkappa \right).$$

*Proof.* From Theorem 3.2, there exists three positive constants $\mathfrak{d}_r(\theta, \rho, n, r, p, q)$, $C(\Omega, n, r, p, q, \varkappa)$,

and $K\left(\lceil l\rceil - l, q\right) \geq 1$, cf. (3.10), such that for $d \leq \mathfrak{d}_r$ and $\eta = 0, \ldots, \lfloor l \rfloor$

$$\begin{aligned} |u|_{\eta,q,\Omega} &\leq C \cdot K \Big( d^{r-l-n/p+n/q} \, |u|_{r-l+\eta,p,\Omega} \\ &\qquad + d^{n/\gamma+(\eta+\mu-\lfloor l\rfloor)_+-\eta} \left\| \prod_{|\alpha|=(\eta+\mu-\lfloor l\rfloor)_+} \partial^\alpha u|_A \right\|_{\varkappa} \Big) \end{aligned}$$

and for $\eta = l$ that

$$|u|_{l,q,\Omega} \leq C \cdot K \left( d^{r-l-n/p+n/q} \, |u|_{r,p,\Omega} + d^{n/\gamma+\mu-l} \left\| \prod_{|\alpha|=\mu} \partial^\alpha u|_A \right\|_{\varkappa} \right).$$

We have taken the constants to be the minimum or maximum over $\eta = 0, \ldots, \lfloor l \rfloor, l$ as required. Additionally we have restricted $\mathfrak{d}_r$ to be at most one. For $\eta = 0, \ldots, \lfloor l \rfloor$, we have applied Theorem 3.2 with $r = r - l + \eta$ and $\mu = (\eta + \mu - \lfloor l \rfloor)_+$, introducing a dependence of $\mathfrak{d}_r$ on $p$ and $q$ through $l$, which along with $n$ and $r$ has substituted for the dependence on $\mu$. It also follows for all $\eta = 0, \ldots, \lfloor l \rfloor$ that

$$\left\| \prod_{|\alpha|=(\eta+\mu-\lfloor l\rfloor)_+} \partial^\alpha u|_A \right\|_{\varkappa} \leq \left\| \prod_{|\alpha|\leq\mu} \partial^\alpha u|_A \right\|_{\varkappa}, \tag{3.13}$$

with a similar bound holding for $\left\| \prod_{|\alpha|=\mu} \partial^\alpha u|_A \right\|_{\varkappa}$. It follows from

$$(\eta + \mu - \lfloor l \rfloor)_+ - \eta \geq \mu - \lfloor l \rfloor \geq \mu - l$$

and $\mathfrak{d}_r \leq 1$ that for all $d \leq \mathfrak{d}_r$,

$$d^{n/\gamma+(\eta+\mu-\lfloor l\rfloor)_+-\eta} \leq d^{n/\gamma+\mu-l}. \tag{3.14}$$

It follow from $r-l \in \mathbb{N}$ that for each $\eta = 0, \ldots, \lfloor l \rfloor$ we have $r-l+\eta \in \mathbb{N}$ and $0 \leq r-l+\eta \leq r-l$, which implies that

$$|u|_{r-l+\eta,p,\Omega} \leq \|u\|_{r,p,\Omega}. \tag{3.15}$$

Combining the preceding bounds we obtain for all $d \leq \mathfrak{d}_r$ and $\eta = 0, \ldots, \lfloor l \rfloor, l$ that

$$|u|_{\eta,q,\Omega} \leq C \cdot K \left( d^{r-l-n/p+n/q} \|u\|_{r,p,\Omega} + d^{n/\gamma+\mu-l} \left\| \prod_{|\alpha|\leq\mu} \partial^\alpha u|_A \right\|_\varkappa \right). \tag{3.16}$$

If $q = \infty$ then the result follows immediately. If $1 \leq q < \infty$ it follows from (3.16) that

$$\begin{aligned} \|u\|_{l,q,\Omega} \quad \leq \quad & C \cdot K \cdot (\lfloor l \rfloor + 2)^{1/q} \Big( d^{r-l-n/p+n/q} \|u\|_{r,p,\Omega} \\ & + d^{n/\gamma+\mu-l} \left\| \prod_{|\alpha|\leq\mu} \partial^\alpha u|_A \right\|_\varkappa \Big). \end{aligned}$$

The result then follows by incorporating the constant $(\lfloor l \rfloor + 2)^{1/q}$ into $C$, with the dependence on $\lfloor l \rfloor$ substituted with $n, r, p$ and $q$. □

Only the case that $p = q = \varkappa = 2$ will be used in Section 3.3.

**Corollary 3.** Let $r$ be a real number, $\mu$ be a nonnegative integer such that $r - \mu > n/2$. Then, there exist three positive constants $\mathfrak{d}_r$ (dependent on $\theta, \rho, n$ and $r$), $C$ (dependent on $\Omega, n$ and $r$), and $K$ (explicitly dependent on $\lceil l \rceil - l$, cf. (3.10) with $q = 2$) satisfying the following property: for any set $A \subset \overline{\Omega}$, such that $d = \delta\left(A, \overline{\Omega}\right) \leq \mathfrak{d}_r$, $u \in W^{r,2}(\Omega)$ and real number $l \in [0, r-\mu]$ such that $r - l \in \mathbb{N}$,

$$\|u\|_{l,2,\Omega} \leq C \cdot K \left( d^{r-l} \|u\|_{r,2,\Omega} + d^{n/2+\mu-l} \left\| \prod_{|\alpha|\leq\mu} \partial^\alpha u|_A \right\|_2 \right). \tag{3.17}$$

## 3.3 Application: Unsymmetric Meshless Methods for Operator Equations

In this section, only the dependence of constants on the discretization parameters $r$ and $s$ is explicitly stated since we regard the spaces and mappings involved as fixed.

We now apply the sampling inequality stated in Corollary 3 to Schaback's framework for unsymmetric meshless methods for operator equations [63]. Due to unaddressed errors and inconsistencies contained in its original formulation, we provide a reformulation. On a technical level, it differs substantially, e.g., certain spaces have been eliminated, the inequalities apply over possibly different spaces, the proof of the error bound has been slightly modified, and the convergence results are different. Despite these changes, the underlying ideas of this framework are the same and entirely due to Schaback [63].

The framework provides an error bound for meshless methods which approximately solve a linear operator equation in the following setting. The first requirement is a continuous and bijective linear operator $L : U \to F$ mapping from the *solution space* to the *data space*. The spaces $U$ and $F$ are assumed to be complete in order to ensure the boundedness of $L^{-1} : F \to U$. It is also assumed that the exact solution $u^* \in \tilde{U}$ where $\tilde{U} \subset U$ is called the *regularity subspace*. We denote $\tilde{F} := L\tilde{U}$. The framework requires a scale of finite-dimensional trial subspaces $U_r \subset \tilde{U}$ equipped with a projector $\Pi_r : \tilde{U} \to U_r$. The framework requires a linear, continuous, and bijective *test mapping* $\Lambda : F \to T$, where the *test space* $T$ is assumed to be complete in order to ensure the boundedness of $\Lambda^{-1}$. We denote $\tilde{T} := \Lambda\tilde{F}$. Test data from $T$ is discretized into finite-dimensional test subspaces $T_s$ with a *test discretization* mapping

$$\pi_s : T \to T_s \tag{3.18}$$

the operator norm of which must be bounded by a constant, which is independent of $s$. It follows that the operator norm of

$$\pi_s \Lambda L : U \to T_s$$

is bounded similarly since

$$\|\pi_s \Lambda L\|_{U \to T_s} \leq \|\pi_s\|_{T \to T_s} \|\Lambda L\|_{U \to T}$$

In order to apply the error bound of Schaback's framework a number of inequalities must be supplied. The first of these is the *trial space approximation property*

$$\|u - \Pi_r u\|_U \leq \epsilon(r) \|u\|_{\tilde{U}} \text{ for all } u \in \tilde{U}. \tag{3.19}$$

The second inequality is the test discretization's *stability condition*

$$\|\Lambda L u_r\|_T \leq \beta(s) \|\pi_s \Lambda L u_r\|_{T_s} \text{ for all } u_r \in U_r. \tag{3.20}$$

If the *stability factor* $\beta(s)$ grows as the test discretization is refined, i.e., as $s$ decreases towards zero, then the order of convergence in the final error bound (3.23) will be less than that provided by the trial space approximation property (3.19). When the stability factor does not grow, the test discretization is called *uniformly stable*. The final inequality required by Schaback's framework involves a numerical method capable of providing an approximate solution $u^*_{r,s} \in U_r$ which satisfies the *numerical method approximation property*

$$\left\|\pi_s \Lambda \left(L u^*_{r,s} - f\right)\right\|_{T_s} \leq C \|\pi_s \Lambda L\|_{U \to T_s} \epsilon(r) \|u^*\|_{\tilde{U}}. \tag{3.21}$$

In particular, if the numerical method computes $u^*_{r,s} \in U_r$ which minimizes the left hand side of (3.21) then the constant is at most one, since

$$\begin{aligned} \left\|\pi_s \Lambda \left(L u^*_{r,s} - f\right)\right\|_{T_s} &\leq \|\pi_s \Lambda L \left(\Pi_r u^* - u^*\right)\|_{T_s} \\ &\leq \|\pi_s \Lambda L\|_{U \to T_s} \|\Pi_r u^* - u^*\|_U \\ &\leq \|\pi_s \Lambda L\|_{U \to T_s} \epsilon(r) \|u^*\|_{\tilde{U}}. \end{aligned} \tag{3.22}$$

**Theorem 3.3.** [63, Theorem 1] Given the setting stated above, if the inequalities (3.19), (3.20), and (3.21) are satisfied then the following error bound holds:

$$\left\|u^* - u^*_{r,s}\right\|_U \leq \left(1 + \beta(s)\left\|(\Lambda L)^{-1}\right\|_{T\to U}\|\pi_s \Lambda L\|_{U\to T_s}(1+C)\right)\epsilon(r)\|u^*\|_{\tilde{U}}.$$

*Proof.* We have that

$$\begin{aligned}\left\|u^* - u^*_{r,s}\right\|_U &\leq \|u^* - \Pi_r u^*\|_U + \left\|\Pi_r u^* - u^*_{r,s}\right\|_U \\ &\leq \epsilon(r)\|u^*\|_{\tilde{U}} + \left\|\Pi_r u^* - u^*_{r,s}\right\|_U\end{aligned}$$

$$\begin{aligned}\left\|\Pi_r u^* - u^*_{r,s}\right\|_U &\leq \left\|(\Lambda L)^{-1}\right\|_{T\to U}\left\|\Lambda L\left(\Pi_r u^* - u^*_{r,s}\right)\right\|_T \\ &\leq \beta(s)\left\|(\Lambda L)^{-1}\right\|_{T\to U}\left\|\pi_s \Lambda L\left(\Pi_r u^* - u^*_{r,s}\right)\right\|_{T_s} \\ &\leq \beta(s)\left\|(\Lambda L)^{-1}\right\|_{T\to U}\Big(\|\pi_s \Lambda L(\Pi_r u^* - u^*)\|_{T_s} \\ &\qquad + \left\|\pi_s \Lambda L\left(u^* - u^*_{r,s}\right)\right\|_{T_s}\Big) \\ &= \beta(s)\left\|(\Lambda L)^{-1}\right\|_{T\to U}\|\pi_s \Lambda L\|_{U\to T_s}\epsilon(r)\|u^*\|_{\tilde{U}}(1+C)\end{aligned}$$

$$\begin{aligned}\left\|u^* - u^*_{r,s}\right\|_U &\leq \epsilon(r)\|u^*\|_{\tilde{U}} \\ &\quad + \beta(s)\left\|(\Lambda L)^{-1}\right\|_{T\to U}\|\pi_s \Lambda L\|_{U\to T_s}\epsilon(r)\|u^*\|_{\tilde{U}}(1+C).\end{aligned}$$

□

The stability condition (3.20) can be established using an *inverse estimate*

$$\|u_r\|_{\tilde{U}} \leq \gamma(r) \|u_r\|_U \text{ for all } u_r \in U_r, \tag{3.23}$$

a *sampling inequality*

$$\|f\|_T \leq C\left(\alpha(s) \|f\|_{\tilde{T}} + \beta(s) \|\pi_s f\|_{T_s}\right) \text{ for all } f \in \tilde{T}, \tag{3.24}$$

and ensuring that a *fine enough test discretization* is chosen such that

$$C\alpha(s)\gamma(r) \|\Lambda L\|_{\tilde{U}\to\tilde{T}} \left\|(\Lambda L)^{-1}\right\|_{T\to U} \leq \frac{1}{2}, \tag{3.25}$$

where $C$ is the constant appearing in (3.24). Typically, $\gamma(r) \to \infty$ as $r \to 0$, while $\alpha(s) \to 0$ as $s \to 0$.

**Proposition 6.** *[63, Theorem 2] If* (3.23), (3.24), *and* (3.25) *hold then so does* (3.20).

*Proof.* We have that

$$\begin{aligned}
\|\Lambda L u_r\|_T &\leq C\left(\alpha(s) \|\Lambda L u_r\|_{\tilde{T}} + \beta(s) \|\pi_s \Lambda L u_r\|_{T_s}\right) \\
&\leq C\left(\alpha(s) \|\Lambda L\|_{\tilde{U}\to\tilde{T}} \|u_r\|_{\tilde{U}} + \beta(s) \|\pi_s \Lambda L u_r\|_{T_s}\right) \\
&\leq C\left(\alpha(s) \|\Lambda L\|_{\tilde{U}\to\tilde{T}} \gamma(r) \|u_r\|_U + \beta(s) \|\pi_s \Lambda L u_r\|_{T_s}\right) \\
&\leq C\Big(\alpha(s) \|\Lambda L\|_{\tilde{U}\to\tilde{T}} \gamma(r) \left\|(\Lambda L)^{-1}\right\|_{T\to U} \|\Lambda L u_r\|_T \\
&\qquad + \beta(s) \|\pi_s \Lambda L u_r\|_{T_s}\Big) \\
&\leq \frac{1}{2} \|\Lambda L u_r\|_T + C\beta(s) \|\pi_s \Lambda L u_r\|_{T_s}
\end{aligned}$$

and the result follows by incorporating the constant $2C$ in $\beta(s)$. □

### 3.3.1 Convergence Results for the Poisson Problem

We consider the example from [63, Section 4.1], a Poisson problem with mixed, inhomogeneous boundary data: let $\Omega$ be a bounded domain in $\mathbb{R}^n$ with a Lipschitz-continuous boundary. Suppose that $\Gamma^N$ and $\Gamma^D$ are connected, mutually disjoint subsets of $\partial\Omega$ such that $\overline{\Gamma^N \cup \Gamma^D} = \partial\Omega$. We denote $\Omega_1 := \Omega$, $\Omega_2 := \Gamma^D$, and $\Omega_3 = \Gamma^N$ so that the dimension of each domain is given by $n_1 = n$, and $n_2, n_3 = n - 1$. Let $m, \tilde{m}$ be nonnegative real numbers such that $\tilde{m} - m \in \mathbb{N}$, and

$$\begin{aligned}
(m_1, m_2, m_3) &:= (m, m + 3/2, m + 1/2) \\
U &:= H^{m+2}(\Omega) \\
F &:= F^1 \times F^2 \times F^3 \\
&:= H^{m_1}(\Omega_1) \times H^{m_2}(\Omega_2) \times H^{m_3}(\Omega_3) \\
Lu &:= \left( -\Delta u, u|_{\Gamma^D}, \frac{\partial u}{\partial n}|_{\Gamma^N} \right),
\end{aligned} \tag{3.26}$$

with analogous definitions made for $(\tilde{m}_1, \tilde{m}_2, \tilde{m}_3)$, $\tilde{U}$, and $\tilde{F}$. With the space $F$ equipped with the norm $\|\cdot\|_F^2 := \|\cdot\|_{F^1}^2 + \|\cdot\|_{F^2}^2 + \|\cdot\|_{F^3}^2$, it follows that the linear operator $L$, as defined above, is continuously invertible either as $L : U \to F$ or $L : \tilde{U} \to \tilde{F}$.

We assume that the solution comes from $\tilde{U}$ and that the trial space $U_r$ is chosen such that the trial space approximation property (3.19) holds with $\epsilon(r) = O\left(r^{\tilde{m}-m}\right)$, a property satisfied by kernel-based meshless trial spaces, c.f. Narcowich et al. [50, 51], and finite-element trial spaces [10, Theorem 4.5.11]. We also assume that the inverse estimate (3.23) holds with $\gamma(r) = O\left(r^{m-\tilde{m}}\right)$, as is the case for finite-element trial spaces [10, Theorem 4.4.20]. Obtaining an inverse estimate with the expected factor $\gamma(r) = O\left(r^{m-\tilde{m}}\right)$ appears to be an open problem for kernel-based meshless trial spaces. Narcowich et al. [51] provide an inverse estimate with the expected factor for the case of Sobolev spaces over $\mathbb{R}^n$. Both

Schaback and Wendland [61], and Duan [18] provide inverse estimates for Sobolev spaces over a domain. Unfortunately, the factor involved in these inverse estimates are worse than the finite-element case. Further progress on this problem is expected to be reported in the thesis of Rieger [58].

We consider the case of strong testing here, which means that the test mapping $\Lambda : F \to T$ is just the identity mapping and that each test space $T^k$ coincides with the corresponding data space $F^k$. Weak testing is also possible, in which case the test functionals integrate functions in $F^k$ against test functions, resulting in the test data in each $T^k$ acquiring additional smoothness. This is discussed in detail by Schaback [63, 65]. Each domain is discretized onto finite subsets $Y_s^k \subset \Omega_k$, with the same fill distance $s = \delta\left(Y_s^k, \Omega_k\right)$. Furthermore, they are assumed to satisfy the property that $\#Y_s^k$ is bounded by $s^{-n_k}$ up to a constant, as is the case for domain discretization with a uniformly bounded mesh ratio [63]. We define discrete test spaces

$$T_s^k := \mathbb{R}^{\#\{\alpha : |\alpha| \leq \mu_k\} \cdot \#Y_s^k}$$

equipped with a norm

$$\|\cdot\|_{T_s^k} := s^{n_k/2} \|\cdot\|_2 \tag{3.27}$$

and a test discretization $\pi_s^k : T^k \to T_s^k$

$$\pi_s^k f_k := \prod_{|\alpha| \leq \mu_k} \partial^\alpha f_k|_{Y_s^k} \text{ for all } f_k \in T^k \tag{3.28}$$

where $\mu_k$ is an integer such that $m_k - \mu_k - n_k/2 > 0$, and furthermore this difference is independent of $k$. The discrete test space $T_s := T_s^1 \times T_s^2 \times T_s^3$ is defined and equipped with a norm, analogously to $F$ and $T$. The test space $T$ is then equipped with a test discretization

$\pi_s : T \to T_s$ defined by

$$\pi_s f := \left(\pi_s^1 f_1, \pi_s^2 f_2, \pi_s^3 f_3\right) \text{ for all } f = (f_1, f_2, f_3) \in T, \tag{3.29}$$

**Proposition 7.** *If for each $k$, $m_k - \mu_k - n_k/2 > 0$ then $\pi_s : T \to T_s$ is well-defined and the operator norm $\|\pi_s\|_{T \to T_s}$ is bounded independently of $s$.*

*Proof.* Suppose $f = (f_1, f_2, f_3) \in T$. Since $m_k - \mu_k > n_k/2$ we have from the Sobolev embedding theorem, Theorem 1.3 that $T^k \hookrightarrow C^{\mu_k}\left(\overline{\Omega_k}\right)$ and therefore the test discretization is both well-defined and there exists some constant independent of $f = (f_1, f_2, f_3)$ such that for each $f_k$,

$$\|f_k\|_{C^{\mu_k}\left(\overline{\Omega_k}\right)} \leq C \|f_k\|_{T^k}.$$

Since $\# Y_s^k$ is bounded by $s^{-n_k}$ up to some constant which is independent of $s$, it follows that

$$\begin{aligned}
\|\pi_s f\|_{T_s}^2 &= \sum_{k=1}^{3} \left\|\pi_s^k f_k\right\|_{T_s^k}^2 \\
&= \sum_{k=1}^{3} s^{n_k} \sum_{x \in Y_s^k} \sum_{|\alpha| \leq \mu_k} \left|\partial^\alpha f(x)\right|^2 \\
&\leq \sum_{k=1}^{3} s^{n_k} \|f\|_{C^{\mu_k}\left(\overline{\Omega_k}\right)}^2 \# \{\alpha : |\alpha| \leq \mu_k\} \cdot \# Y_s^k \\
&\leq C \sum_{k=1}^{3} \|f\|_{T^k}^2 = C \|f\|_T^2.
\end{aligned}$$

□

**Proposition 8.** *There exists a constant $s_0$ such that for all $s \leq s_0$ a sampling inequality* (3.24) *holds with a constant $C$ for the test space $T$ and test discretization $\pi_s : T \to T_s$*

*with* $\alpha(s) := s^{\tilde{m}-m}$, *and* $\beta(s) := s^{\mu_1-m_1} = s^{\mu_2-1/2-m_2} = s^{\mu_3-1/2-m_3}$.

*Proof.* From Corollary 3 and (3.27), it follows that for each $k$ there exist constants $C_k$ and $s_k$ such that for $s \leq s_0 := \min(1, s_1, s_2, s_3)$,

$$\begin{aligned} \|f_k\|_{T^k} &\leq C_k \left( \alpha(s) \|f_k\|_{\tilde{T}^k} + s^{\mu_k - m_k} \left\| \pi_s^k f_k \right\|_{T_s^k} \right) \\ &\leq C_k \left( \alpha(s) \|f_k\|_{\tilde{T}^k} + s^{\mu_1 - m_1} \left\| \pi_s^k f_k \right\|_{T_s^k} \right) \end{aligned}$$

since $\mu_1 - m_1 \leq \mu_2 - m_2 = \mu_3 - m_3$. The result then follows with a constant $C$ by combining the preceding inequalities. □

We assume that the $s$ is sufficiently small to satisfy the requirements of Proposition 8 and (3.25). Even in the fractional case, the sampling inequality introduced here provides $\alpha(s) = s^{\tilde{m}-m}$ which shrinks as rapidly as the expected inverse estimate factor $\gamma(r) = r^{m-\tilde{m}}$ grows and thus $s$ and $r$ can be kept proportional. This is in contrast to previous sampling inequalities which necessarily introduce a factor $\alpha(s) = s^{\tilde{m}-\lceil m \rceil}$ when bounding fractional order Sobolev norms, requiring the test discretization to be refined more rapidly than the trial discretization and thus diminishing the order of convergence by $\lceil m \rceil - m$. If the function $u_{r,s}^* \in U_r$ which minimizes the left hand side of (3.21) has been computed, then Schaback's framework provides the error bound (3.23) with constant $C = 1$. The order of convergence established by this error bound, in terms of both the trial and test discretization, is then given by $\beta(h)\,\epsilon(h)$ and satisfies

$$\beta(h)\,\epsilon(h) = O\left( h^{(\tilde{m}-m)+(\mu_1-m_1)} \right).$$

Table 3.1 states particular convergence results using various Sobolev norms and test discretizations in two- and three-dimensions. These results show that, for convergence in higher order norms, the highest order of convergence is obtained using a higher order test

Table 3.1: Order of convergence in various Sobolev norms established by a modified formulation of Schaback's framework, using trial spaces with optimal properties and strong testing with various order test discretizations to solve two- or three-dimensional Poisson problems.

| $U = H^{m+2}(\Omega) =$ | $H^0(\Omega)$ | $H^4(\Omega)$ | $H^5(\Omega)$ | $H^6(\Omega)$ |
|---|---|---|---|---|
| $\mu_1 = 0$ | None | $\tilde{m} - m - 2$ | $\tilde{m} - m - 3$ | $\tilde{m} - m - 4$ |
| $\mu_1 = 1$ | None | None | $\tilde{m} - m - 2$ | $\tilde{m} - m - 3$ |
| $\mu_1 = 2$ | None | None | None | $\tilde{m} - m - 2$ |
| $\mu_1 = 3$ | None | None | None | None |

discretization introduced here.

We note that to obtain a uniformly stable test discretization with $\beta(s) = 1$ would require choosing the order $\mu_k$ of each test discretization to be equal to that of the order $m_k$ of the test space. Unfortunately, this does not seem to be possible: in order for the test discretizations operator norm to be bounded independently of $s$, the order of each test space $T^k$ is required to be greater than that of the test discretization $\mu_k$ by at least $n_k/2$. It follows that the order of convergence, in terms of both the trial and test discretization, provided by this modified formulation of Schaback's framework is always less than that of the trial space approximation property. Another consequence is that the order of $U$ must be at least $2 + n/2$, and therefore convergence in the $L^2$ norm can only be concluded from convergence results in higher order Sobolev norms, using strong testing in this modified formulation of Schaback's framework.

## 3.4 Conclusions

We have further generalized the sampling inequalities of Arcangéli et al. [2], Madych [44], and Wendland and Rieger [73], to optimally bound fractional order Sobolev semi-norms, and to incorporate higher order data into the bound. When used in a modified formulation of Schaback's framework to prove convergence rates for unsymmetric meshless methods this new sampling inequality has two benefits:

1. It results in more optimal estimates for problems involving fractional order Sobolev

spaces, particularly by providing a more optimal constant $\alpha(s)$.

2. For convergence in higher order Sobolev norms, higher order results are obtained using a higher order test discretization in comparison to the zero order test discretization.

Previous work indicates that using unsymmetric collocation to solve the Poisson problem has the same order of convergence as interpolation. It follows that the results presented here may be suboptimal. Therefore, future work may investigate a framework for unsymmetric collocation which exploits stronger assumptions than well-posedness.

# Chapter 4: Visualization Using Fourier Volume Rendering

## 4.1 Introduction

Volume visualization is an important tool for understanding three-dimensional simulation data. Fourier volume rendering (FVR) [20, 41, 45] is a particular volume visualization technique. Previous work on FVR has so far only adapted this technique to deal directly with regular grid data or wavelet data [31], including an indirect adaptation to irregularly sampled data [67]. By indirect it is meant that the data is first sampled onto a three-dimensional grid before volume visualization is performed. As pointed out by others [14, 37, 55, 56], in the context of meshless data this could result in the loss of important detail at feasible grid resolutions, and furthermore would leave meshless data less efficient to visualize than grid data. The purpose of this work is to avoid such issues and adapt FVR to deal directly with a general class of meshless data in the form of a summation of $N$ integrable functions $\Phi_k : \mathbb{R}^3 \to \mathbb{R}$ with coefficients $\alpha_k \in \mathbb{R}$

$$s(x) = \sum_{k=1}^{N} \alpha_k \Phi_k(x). \tag{4.1}$$

## 4.2 Related work

Most related to this work are techniques which also deal directly with meshless data. One such technique is direct slice-based volume visualization which was considered in the context of meshless data in [38] with further improvements made in [72]. The technique presented in those works for visualizing meshless data, in particular radial basis function interpolants, involves sampling the meshless data over an array of two-dimensional sampling planes in a

view-dependent manner. They use a spatial data structure to only evaluate terms whose support is relevant to regions of a given slice. While this is not required, a loss of performance is to be expected otherwise. When visualizing static data, building this data structure can be treated as a pre-processing step. However, this is not always possible, such as when the meshless data is time-varying with moving data sites. In contrast, FVR requires no spatial data structure since it deals with the data's Fourier transform which tends to be centered at the origin. A related technique for slice-based isosurface visualization of meshless data is presented in [15].

Another such technique is splatting. This technique was considered in the context of meshless data in [56]. When using an orthogonal projection, splatting attempts to compute the same image as does FVR. The difference is that splatting computes the image by directly approximating the integral in the spatial domain. This technique involves computing a so-called footprint using numerical integration, which is accumulated at different points in the image for each term in the meshless data. FVR and splatting can be considered complementary since FVR performs most efficiently for data with low frequency content, typically when each term has large spatial support, while splatting performs most efficiently for data with small spatial support, typically resulting in high frequency content. Also, given a fixed cutoff frequency, FVR becomes cheaper when zooming in, due to the associated larger frequency step size decreasing the required amount of sampling. On the other hand, splatting becomes cheaper when zooming out, due to less resolution being required of the footprint. Relatively recent work on splatting meshless data includes multiresolution [47] and hierarchical [37] techniques for handling enormous meshless data sets, and a technique for splatting meshless data with elliptical basis functions [52].

The traditional computer graphics technique of ray-tracing for visualizing isosurfaces [35] is general enough to deal directly with an arbitrary function, and therefore can deal directly with meshless data. Another approach to isosurface visualization [14], uses local tetrahedrizations to compute an isosurface while avoiding globally sampling the meshless data onto a grid. The SPH visualization code SPLASH [55] is also related to this work. This

code implements a number of visualization techniques which deal directly with meshless data, such as volume visualization using splatting, cross section visualization, and surface visualization based on optical depth.

## 4.3 Direct Adaptation of FVR to Meshless Data

Fourier volume rendering computes from a function $s : \mathbb{R}^3 \to \mathbb{R}$ a two-dimensional image $I : \mathbb{R}^2 \to \mathbb{R}$ defined as

$$I(u, v) = \int_{\mathbb{R}} s(tw_t + uw_u + vw_v)\, dt \tag{4.2}$$

where $\{w_t, w_u, w_v\}$ is an orthonormal basis of $\mathbb{R}^3$. In words, for each point on the image plane spanned by $\{w_u, w_v\}$, FVR integrates the function $s$ through the point along the image plane's normal $w_t$. What distinguishes Fourier volume rendering from other volume visualization techniques is that it computes the image via its Fourier transform [20, 41, 45]

$$\mathcal{F}_2 I(u, v) = \mathcal{F}_3 s(uw_u + vw_v). \tag{4.3}$$

Therefore, to perform Fourier volume rendering with meshless data its Fourier transform is required. For grid-based Fourier volume rendering this is obtained by computing a three-dimensional FFT. For meshless data such an expensive computation is not required, since its Fourier transform can be obtained analytically. The inverse Fourier transform can then be approximated using a two-dimensional inverse FFT.

### 4.3.1 The Fourier Transform of Meshless Data

To adapt FVR, the Fourier transform of the image (4.3) is obtained by applying properties of the Fourier transform. By linearity, the Fourier transform of the meshless data is

$$\mathcal{F}_3 s(f) = \sum_{k=1}^{N} \alpha_k \mathcal{F}_3 \Phi_k(f) \tag{4.4}$$

which reduces the problem to computing the Fourier transform of each function $\Phi_k$. We use the Fourier transform convention

$$\mathcal{F}_3\Phi(f) = \int_{\mathbb{R}^3} \Phi(x)\, e^{-2\pi i x^T f} dx \tag{4.5}$$

which is defined when $\Phi$ is integrable: $\Phi \in L^1(\mathbb{R}^n)$. Therefore if each $\Phi_k$ is integrable then the Fourier transform of the meshless data is defined. An important special case is when each $\Phi_k$ is defined in terms of a radial function $\Phi$, in which case (4.5) specializes to

$$\mathcal{F}_3\Phi(f) = \frac{2}{\|f\|} \int_{\mathbb{R}^+} t\phi(t) \sin(2\pi \|f\| t)\, dt \tag{4.6}$$

where $\Phi(x) = \phi(\|x\|)$ [75, Theorem 5.26].

In Section 4.4 the Fourier transform of $\Phi_k$ is obtained for different types of meshless data. This allows for the direct evaluation of the Fourier transform of the meshless data, and therefore by (4.3) the Fourier transform of the image.

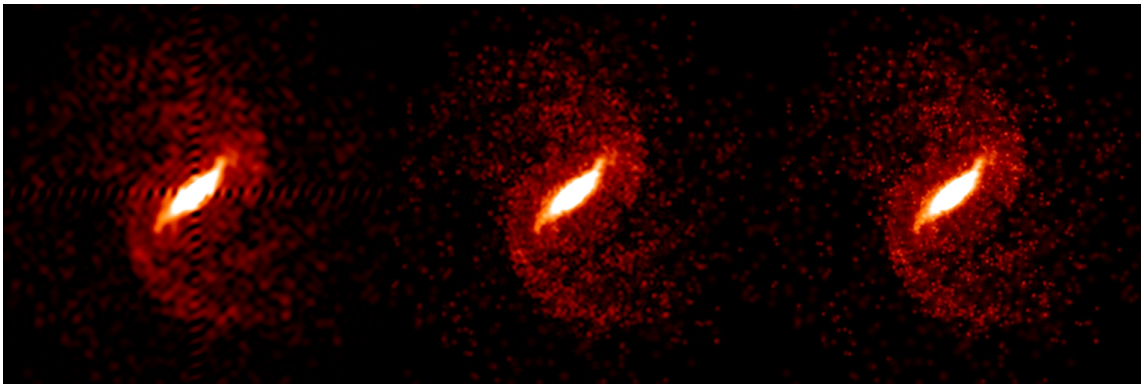
### 4.3.2 Approximation of the Inverse Fourier Transform

Having obtained the Fourier transform of the image, the next step in FVR is to compute the inverse Fourier transform of $\mathcal{F}_2 I$ to obtain the image $I$.

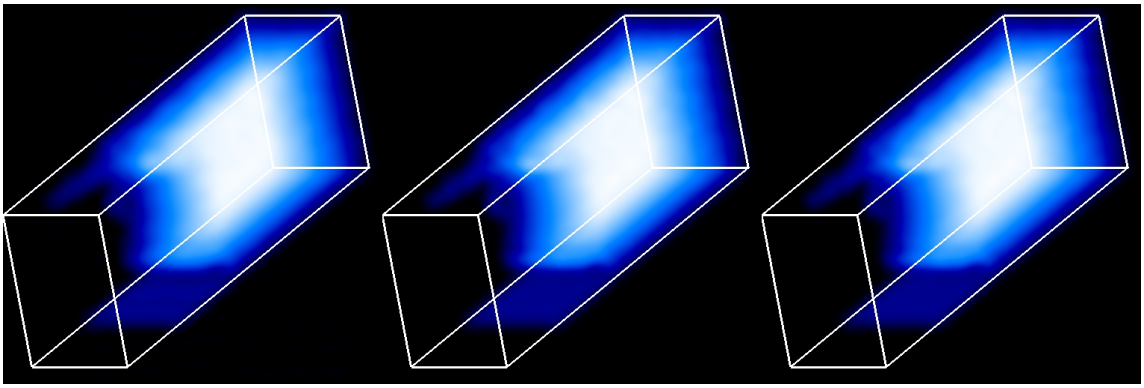
$$I(u,v) = \int_{\mathbb{R}} \int_{\mathbb{R}} \mathcal{F}_2 I(f_u, f_v)\, e^{i2\pi(uf_u + vf_v)} df_u df_v \tag{4.7}$$

For this, a discrete Fourier transform is used

$$I(u,v) \approx \sum_{k_{f_v}=-N_{f_v}}^{N_{f_v}} \sum_{k_{f_u}=-N_{f_u}}^{N_{f_u}} \mathcal{F}_2 I(f_u, f_v)\, e^{i2\pi(uf_u + vf_v)} \Delta_{f_u} \Delta_{f_v} \tag{4.8}$$



(a) The bar galaxy data set (6400 particles) contains significant high frequency detail. Therefore there is a loss of quality when the lowpass filter is made too small as in the left image. However, little ringing is present in the center image, resulting in a nice tradeoff between quality and performance. Here we use an integration step size of 0.065 and 512x512 samples in the spatial domain. From left to right the number of samples and performance in frames per second on an NVIDIA Geforce 8600 GTS is respectively 64x32, 128x64, 192x96, and 63, 21, and 10.4.



(b) The water sloshing in a reservoir data set (11895 particles) contains little high frequency detail. Therefore a lowpass filter can be applied without a noticeable loss of quality. Here we use an integration step size of 0.7 and 512x512 samples in the spatial domain. From left to right the number of samples and performance in frames per second on an NVIDIA Geforce 8600 GTS is respectively 32x16, 64x32, and 128x64, and 92, 40, and 12.4.

Figure 4.1: The above images illustrate the trade-off in performance and quality using the approximation (4.8).

where $u = k_u \Delta_u$, with analogous substitutions made for $v, f_u$ and $f_v$. Also, $k_u$ and $k_v$ vary over $-N_u \ldots N_u - 1$ and $-N_v \ldots N_v - 1$ respectively.

In the language of image processing, by only considering a finite number of samples, an ideal lowpass filter is applied to the image [29, Page 167]. This filter provides a control of performance at the expense of quality. By choosing a sufficiently large cutoff frequency, the effects of the lowpass filter are not apparent. However, choosing a larger cutoff frequency requires more frequency domain sampling, and therefore decreases performance. An insufficiently large cutoff frequency will cause blurring and ringing. While blurring is an acceptable trade-off for increased performance, ringing can be distracting. To decrease the effects of ringing a standard approach is to use a different lowpass filter, such as a Butterworth filter [29, Page 173]. In cases where the image contains low frequency content, a low cutoff frequency can be used without sacrificing image quality. In Figure 4.1 the effects of ringing and blurring are shown for the bar galaxy data set, while for the sloshing data set it is shown that for almost no loss of quality significant gains in performance can be made. This demonstrates that FVR is well suited for data with low frequency content.

Another effect of this approximation is that the image becomes periodic, due to the use of numerical integration with a finite step size in each dimension. This can lead to aliasing if the period is not sufficiently large. However, one can choose the step size, so it can be left up to the user how much aliasing if any at all can be tolerated. If the functions $\Phi_k$ have global support, then there will always be some amount of aliasing. However, these functions are required to be integrable and therefore decreasing towards zero, and so even in this case where aliasing is theoretically unavoidable, its effect can be made negligible, by sampling up to a sufficiently high frequency.

## 4.4 Particular Meshless Methods

In the preceding section the specific form of $\Phi_k$ was not considered, which is necessary for determining its Fourier transform. In this section the form $\Phi_k$ takes for various types of meshless data is considered, from which its Fourier transform is obtained. In each case

Table 4.1: A list of three-dimensional Fourier transforms of various integrable functions used in meshless methods. With the exception of that of the Gaussian, the inverse multiquadric, and the Sobolev spline [75, Theorems 6.10, 6.13, and Page 133], the Fourier transform of each function was computed using (4.6).

| Name | $\Phi(x)=\phi(\|x\|), r=\|x\|$ | $\mathcal{F}\Phi(f), r=\pi\|f\|, m=\pi r$ |
|---|---|---|
| Gaussian | $e^{-\alpha r^2}, \alpha>0$ | $\left(\frac{\pi}{\alpha}\right)^{\frac{3}{2}} e^{-\frac{m^2}{\alpha}}$ |
| Inverse Multiquadric | $\left(c^2+r^2\right)^{-\beta}, c>0, \beta>\frac{3}{2}$ | $\frac{2\pi^{3/2}}{\Gamma(\beta)}\left(\frac{m}{c}\right)^{\beta-3/2} K_{3/2-\beta}(2cm)$ |
| Monaghan's $M_4$ | $(2-r)^3-4(1-r)^3, 0\le r<1$<br>$(2-r)^3, 1\le r<2$<br>$0, r\ge 2$ | $\frac{3\pi}{m^6}(\cos 2m-1)(\cos 2m+m\sin 2m-1)$ |
| Sobolev | $\frac{2^{1-\beta}}{\Gamma(\beta)} r^{\beta-3/2} K_{3/2-\beta}(r), \beta>\frac{3}{2}$ | $(2\pi)^{3/2}\left(1+4m^2\right)^{-\beta}$ |
| Wendland's $\phi_{3,0}$ | $(1-r)_+^2$ | $\frac{\pi}{2m^5}\left(m\cos 2m-\frac{3}{2}\sin 2m+2m\right)$ |
| Wendland's $\phi_{3,1}$ | $(1-r)_+^4(4r+1)$ | $\frac{15\pi}{2m^8}\left(\frac{9}{2}m\sin 2m+\left(6-m^2\right)\cos 2m+4m^2-6\right)$ |
| Wendland's $\phi_{3,2}$ | $(1-r)_+^6\left(35r^2+18r+3\right)$ | $\frac{315\pi}{2m^{11}}\begin{pmatrix}\left(\frac{315}{2}-36m^2\right)\sin 2m\\ +\left(4m^2-123\right)m\cos 2m\\ +32m\left(m^2-6\right)\end{pmatrix}$ |
| Wendland's $\phi_{3,3}$ | $(1-r)_+^8\left(32r^3+25r^2+8r+1\right)$ | $\frac{10395\pi}{4m^{14}}\begin{pmatrix}\left(60m^3-\frac{2295}{2}m\right)\sin 2m\\ -\left(4m^4+1440-375m^2\right)\cos 2m\\ +1440-960m^2+64m^4\end{pmatrix}$ |

an explicit form of the Fourier transform $\mathcal{F}_3\Phi_k$ is given. With this information the image approximation (4.8) can be sampled directly.

### 4.4.1 Kansa's Method

Kansa's method [39,64] generalizes radial basis function interpolation [11,75] to solve partial differential equations using collocation. The resulting meshless data is of the form of (4.1) with

$$\Phi_k = \Phi\left(\cdot - x_k\right)$$

where $\Phi$ is some function, usually referred to as a radial basis function. Assuming $\Phi$ is integrable, each $\Phi_k$ is as well. It follows by the shift property [75, Theorem 5.16] that

$$\mathcal{F}_3\Phi_k\left(f\right) = e^{-i2\pi x_k^T f}\mathcal{F}_3\Phi\left(f\right).$$

In fact, the Fourier transform of the meshless data can be simplified since $\mathcal{F}_3\Phi$ can be factored outside of the summation

$$\mathcal{F}_3 s\left(f\right) = \mathcal{F}_3\Phi\left(f\right)\sum_{k=1}^{N}\alpha_k e^{-i2\pi x_k^T f}. \tag{4.9}$$

Examples of integrable radial basis functions are given in Table 4.1. Non-integrable radial basis functions and polynomial terms are sometimes used, examples of which include the multiquadric and the thin-plate spline. Our adaptation of FVR does not apply to data using such radial basis functions. Splatting is also unable to deal with such radial basis functions for the same reason of non-integrability. While slice-based volume visualization can deal with such radial basis functions, it does so much slower compared to integrable radial basis functions, due to it being required that every term is evaluated at every point, instead of only a few local terms.

### 4.4.2 Meshless Symmetric Collocation

Meshless symmetric collocation is another method for solving partial differential equations, which uses generalized interpolation [23, 26, 28, 75]. For concreteness the Poisson equation with Dirichlet boundary conditions is considered. In this case the functions $\Phi_k$ are

$$\Phi_k\left(x\right) = \begin{cases} \Delta\Phi\left(x - x_k\right), & 1 \le k \le N_L \\ \Phi\left(x - x_k\right), & 1 \le k - N_L \le N_B \end{cases} \tag{4.10}$$

where $x_1 \ldots x_{N_L}$ are in the interior and $x_{N_L+1} \ldots x_{N_L+N_B}$ are on the boundary of the domain. Applying the shift and differentiation properties [75, Theorem 5.16] of the Fourier

transform shows that

$$\mathcal{F}_3\Phi_k(f) = \begin{cases} -(2\pi\|f\|)^2 e^{-i2\pi x_k^T f}\mathcal{F}_3\Phi(f), & 1 \le k \le N_L \\ e^{-i2\pi x_k^T f}\mathcal{F}_3\Phi(f), & 1 \le k - N_L \le N_B \end{cases} \tag{4.11}$$

The factor $\mathcal{F}_3\Phi(f)$ can be factored outside of the summation.

$$\mathcal{F}_3 s(f) = \mathcal{F}_3\Phi(f)\left(-(2\pi\|f\|)^2 \sum_{k=1}^{N_L} \alpha_k e^{-i2\pi x_k^T f} + \sum_{k=N_L+1}^{N_L+N_B} \alpha_k e^{-i2\pi x_k^T f}\right) \tag{4.12}$$

For this problem, the function $\Phi$ should be in $C^4$. For example, the Gaussian or the Wendland function $\phi_{3,2}$ [75, Page 129] could be used. These functions and others appear along with their Fourier transforms in Table 4.1.

### 4.4.3 Smoothed Particle Hydrodynamics

Smoothed particle hydrodynamics (SPH) is a particle-based meshless method for simulating fluid dynamics [49]. Underlying this method is a kernel approximation

$$A_s(x) = \sum_{k=1}^{N} m_k \frac{A_k}{\rho_k} W(x - x_k, h_k) \tag{4.13}$$

where $A$ is a function being approximated such as density, and $A_k, m_k, \rho_k, r_k, h_k$ are respectively the function value, mass, density, position, and smoothing-length of each particle. Letting $\alpha_k = m_k \frac{A_k}{\rho_k}$ and $\Phi_k(x) = W(x - x_k, h_k)$ shows that equation (4.13) is in a form consistent with (4.1). A common choice for the kernel is

$$\Phi_k(x) = \frac{1}{4\pi h_k^3} M_4\left(\frac{\|x - x_k\|}{h_k}\right). \tag{4.14}$$

which, by the shift and scaling properties [75, Theorem 5.16], has a Fourier transform of

$$\mathcal{F}_3\Phi_k(f) = \frac{1}{4\pi}\mathcal{F}_3 M_4(h_k\|f\|)\, e^{-i2\pi x_k^T f} \tag{4.15}$$

The function $M_4$ is listed in Table 4.1.

## 4.5 Implementation

This technique consists of two major computations: the first step is to sample the Fourier transform, and the second step is to apply an inverse FFT to those samples. The Fourier transform sampling is simple to implement since it merely involves sampling each term in a small window around zero, without the use of any complicated data structures. The second step merely involves applying an existing FFT code to the Fourier transform samples. Each of these steps are overall straightforward to implement, but there are a few details of the implementation worth mentioning.

The Fourier transforms of the compactly-supported polynomials are not defined at zero as written and are difficult to sample accurately near zero due to round-off error. One solution to this problem is to compute the limit of these functions at zero using l'Hospital's rule and to then perform linear or quadratic interpolation between zero and points sufficiently far from zero where the function can be sampled without significant round-off error. This approximation is reasonable since the derivatives of these functions are typically quite small near zero.

For many types of meshless data the Fourier transform of the radial basis function used can be factored out of the summation as in (4.9) and (4.12), resulting in less computation per term. For the graphics hardware implementation described in the next section, factoring out the Fourier transform modestly increases the performance results by 15 to 45 percent.

Because the meshless data's Fourier transform is conjugate symmetric it is only required to sample it over a half plane since the rest of the samples can be obtained by complex

conjugation. This halves the amount of sampling required.

### 4.5.1 Implementation on Graphics Hardware

We have implemented this technique on NVIDIA graphics hardware[1] using their CUDA interface[2]. A brief description of this implementation is as follows. The first step, sampling of the Fourier transform, is implemented as a CUDA kernel. This kernel is executed over a grid of threads where each thread is responsible for computing a partial sum of one sample. Within this grid, threads are organized into thread blocks, where each thread within a block has access to fast shared memory. Since each thread accesses the same parameters from memory, if a thread block is of length $N$, then each thread loads into shared memory the parameters of one of $N$ terms. Once all of the parameters are loaded into shared memory, each thread accumulates the $N$ terms before moving on to the next batch of terms. The code implementing this step is listed in Appendix A. After sampling is performed the image is computed from its Fourier transform samples using a complex-to-real FFT provided by the CUDA FFT library.

Table 4.2 gives frame rates achieved for visualizing various data sets. These performance measurements indicate that this technique is quite capable of interactively displaying meshless data, and is scalable with respect to the data size, the number of Fourier transform samples, and available hardware. Furthermore, these results indicate that the 512x512 complex-to-real FFT is not a bottleneck with respect to achieving interactive frame rates.

The code has been released as the open source library *libMeshlessVis* and is available for download[3].

## 4.6 Applications

We have applied this technique to meshless-based simulations in two application domains. It was first used to visualize meshless data generated by MASS99 [1], a code which couples

[1] `http://www.nvidia.com/page/geforce8.html`
[2] `http://developer.nvidia.com/cuda/`
[3] `http://code.google.com/p/libmeshlessvis`

Table 4.2: Performance measured in frames per second when visualizing various data sets interactively. In each case the $M_4$ function is used with an image size of 512x512. The number of partial sums was chosen such that the number of thread blocks was at least the number of multiprocessors.

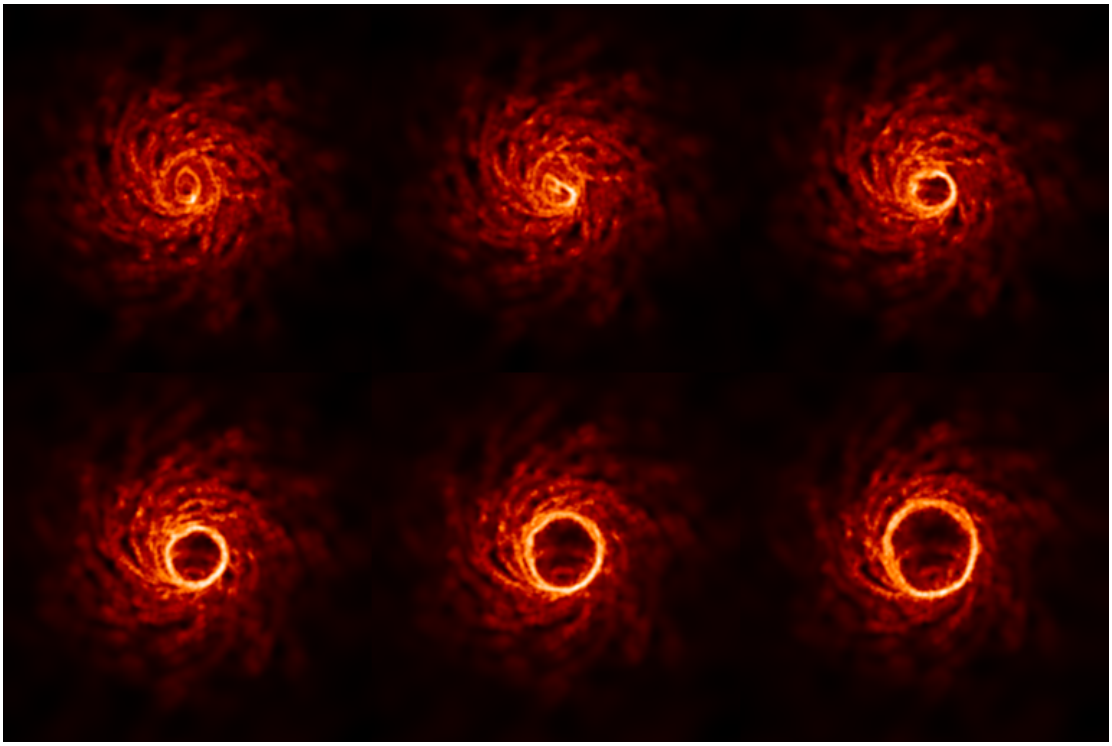
| 8600 GTS | | Number of Fourier transform samples | | | | |
|---|---|---|---|---|---|---|
| Data set | # of terms | 32x16 | 64x32 | 128x64 | 256x128 | 512x256 |
| Ring galaxy | 13107 | 89 | 36 | 11 | 2.8 | 0.8 |
| Bar galaxy | 6400 | 124 | 63 | 21 | 5.8 | 1.4 |
| Sloshing | 11895 | 92 | 40 | 12.4 | 3.6 | 1.2 |
| Cellular Structure | 161973 | 12.8 | 4 | 1.2 | 0.4 | n/a |

| 8800 GTS | | Number of Fourier transform samples | | | | |
|---|---|---|---|---|---|---|
| Data set | # of terms | 32x16 | 64x32 | 128x64 | 256x128 | 512x256 |
| Ring galaxy | 13107 | 170 | 86 | 29 | 8 | 2.4 |
| Bar galaxy | 6400 | 210 | 133 | 54 | 16 | 4.4 |
| Sloshing | 11895 | 178 | 92 | 32 | 9.2 | 2.4 |
| Cellular Structure | 161973 | 30 | 10 | 2.8 | 0.8 | 0.4 |

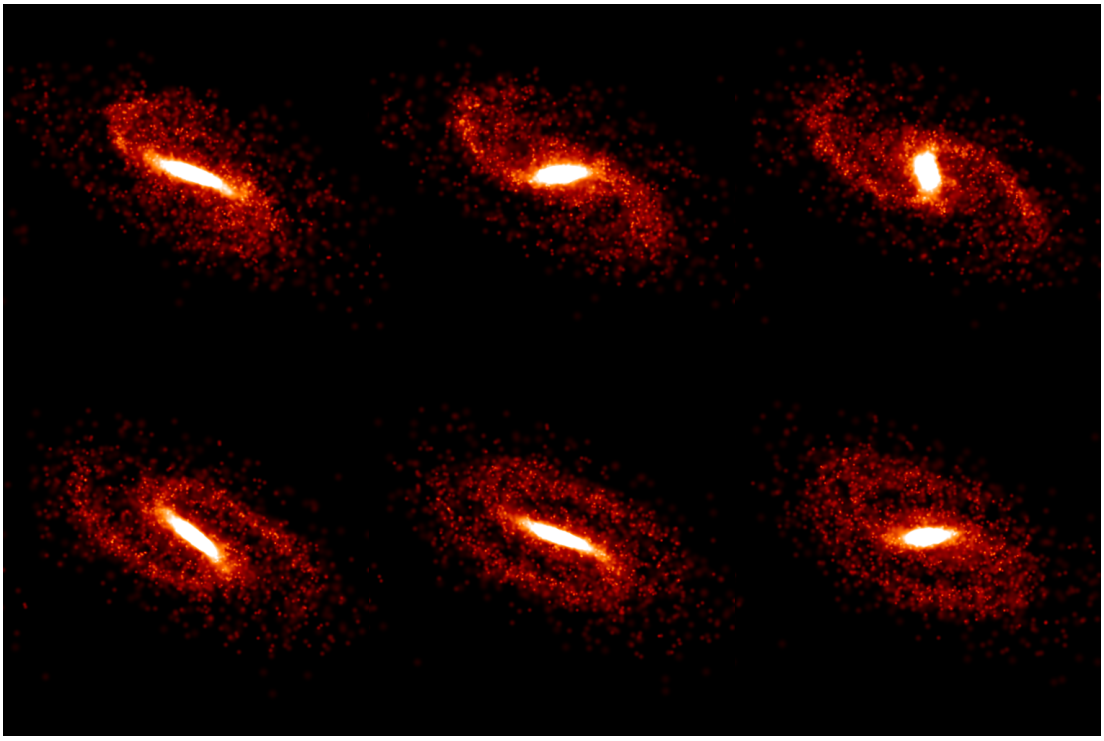
N-body gravity with SPH-based hydrodynamics. It was also used to visualize simulations of fluid sloshing in a reservoir [71] and fluid flow through cellular structure subjected to compressive loading [57, 70]. Further discussion is given by Corrigan et al. [16], where in each case it is observed that Fourier volume rendering produces animations or images with information useful for understanding the simulation.

## 4.7 Future work

We are currently investigating combining Fourier volume rendering and splatting together as a single hybrid technique. This could be done by decomposing the meshless data based on the frequency content of each basis function. In particular, splatting will deal with basis functions with high frequency content (i.e. low spatial support), or that need to be spatially filtered to avoid aliasing, while Fourier volume rendering will deal with low frequency content. Although we have focused on meshless data sets, we are also investigating the application of this approach to unstructured grid FEM data sets, which are also of the form (4.1).

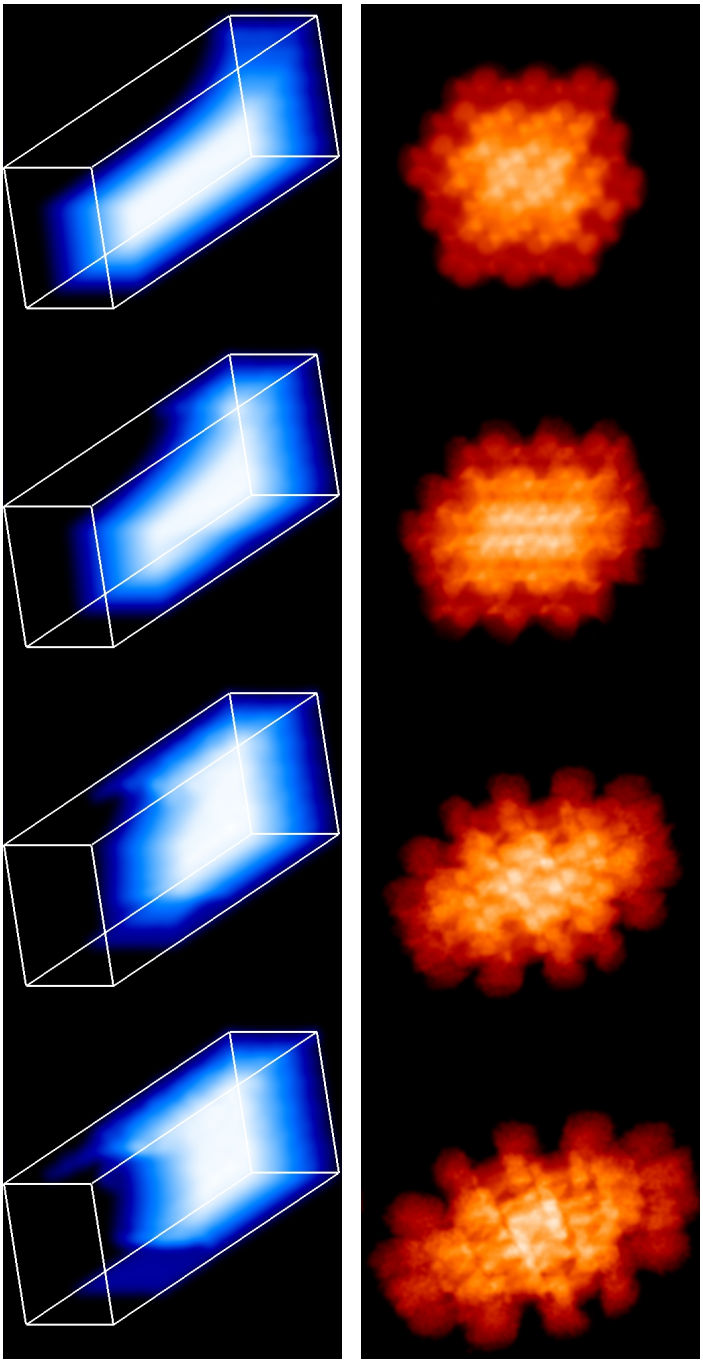


(a) A collisional ring galaxy.



(b) A bar galaxy.

Figure 4.2: Visualization of meshless astrophysical data sets.



(a) Water sloshing in a reservoir. (b) Fluid in a cellular structure.

Figure 4.3: ]
Visualization of meshless fluid dynamics data sets.

Fourier volume rendering has been enhanced to include spatial depth cues and lighting effects [21, 69], which could possibly be used with this adaptation of FVR. We have also left as future work this technique's application to data resulting from meshless methods such as moving least squares [5] where the meshless data is defined at each point by a local least squares fit with respect to a polynomial basis, since it is not clear what the Fourier transform of this type of data is.

## 4.8 Conclusions

We have presented an adaptation of Fourier volume rendering which for the first time enables it to deal directly with meshless data. Because the adaptation is direct, it avoids the prohibitive computational and memory costs and quality loss of discretizing the meshless data into a three-dimensional grid. Other advantages of this adaptation of Fourier volume rendering of meshless data include a trade-off between image quality and performance via lowpass filtering. While lowpass filtering will cause ringing in images with high frequency content, it is otherwise a highly effective approximation. Due to the technique dealing with the Fourier transform of the meshless data, no spatial data structures are needed. Therefore, the technique is indifferent to whether or not the meshless data's geometry is static or dynamic. Because a general form of meshless data was considered this technique is applicable to a number of types of meshless data. Finally, we demonstrated the technique's usefulness in visualizing different meshless-based simulations.

In general we believe that it is important that visualization techniques which deal directly with meshless data are available, so that meshless methods are not at a disadvantage when being considered for use in scientific simulation. Therefore, we encourage further work on adapting visualization techniques to deal directly with meshless data.

# Chapter 5: Running Unstructured Grid Based CFD Solvers on Modern Graphics Hardware

## 5.1 Introduction

Over the past few year graphics hardware (GPU) has seen a tremendous increase in performance, with the latest GeForce 200 series and Tesla 10 series GPUs from NVIDIA now achieving roughly one teraflop of performance, roughly an order of magnitude higher performance than high-end CPUs [53, Sec. 1.2]. In addition to this high computational performance the latest modern graphics hardware offers increasing memory capacity, as well as support for 64-bit floating point arithmetic. Together with CUDA, which exposes the GPU as a general-purpose, parallel multicore processor, the GPU offers tremendous potential for applications in computational fluid dynamics.

In order to fully exploit the computational power of such hardware, considerable care is required in the coding and implementation, particularly in the memory access pattern. CUDA makes available general-purpose global memory, which is not automatically cached and exhibits high latency in comparison with its instruction throughput. Furthermore, with earlier CUDA-enabled GPUs, there were stringent requirements for achieving optimal effective memory bandwidth, with a large loss of performance when these requirements went unmet. With the data-dependent memory access of unstructured grid based solvers, this loss of performance is almost assured. However, with due care, *structured* grid based solvers can meet these requirements due to the regular memory access patterns of such solvers, as described in the work of Brandvik and Pullan [8, 9], and Tölke [68]. So far, the implementation of optimized *unstructured* grid based solvers for modern graphics hardware has been relatively rare, perhaps due to these stringent requirements. An alternative means of accessing GPU memory is via texture memory, which offers automatic caching

intended for memory access patterns which exhibit two-dimensional spatial locality, and has been effectively used, for example, in the CUDA SDK [30]. However, this type of memory is inappropriate for the indirect memory access of three-dimensional unstructured grid solvers. We note that implementing CFD solvers on graphics hardware predates CUDA. In fact, just prior to its first release, Owens et al. [54] comprehensively surveyed the field of general-purpose computation on graphics hardware (GPGPU), which included a number of primarily structured grid based solvers, such as those of Harris [34], Scheidegger et al. [12], and Hagen et al. [32]. However, the architecture has changed substantially and many of the limitations of GPGPU via traditional graphics APIs such as OpenGL are no longer an issue.

The most recent CUDA-enabled GPUs have looser requirements for achieving high effective memory bandwidth. Roughly speaking, memory no longer needs to be accessed in a specific order by consecutive threads. Rather, high effective memory bandwidth can be achieved as long as consecutive threads access nearby locations in memory, which is called *coalescing*. Thus, if an appropriate memory access pattern is obtained, one can expect that modern GPUs will be capable of achieving high effective memory bandwidth and in general high performance for unstructured grid based CFD solvers. The purpose of this work is to study techniques which achieve this.

The remainder of the chapter is organized as follows: Section 5.2 describes the solver considered: a three-dimensional finite volume discretization of the Euler equations for inviscid, compressible flow over an unstructured grid. Section 5.3 considers the techniques used to achieve high performance with modern GPUs for unstructured grid solvers. After giving an overview of the code, techniques are described to achieve a reduction of total memory access by overlapping redundant computation, obtain high effective memory bandwidth using an appropriate numbering scheme as well avoiding divergent branching. This is followed by a discussion of the issue of employing shared memory with unstructured grid solvers. Results are given in Section 5.4. These show the order of magnitude speed-up obtained with modern GPUs in comparison with a parallelized shared-memory OpenMP code running on

a quad-core CPU.

## 5.2 Euler Solver

We consider the Euler equations for inviscid, compressible flow,

$$\frac{d}{dt}\int_{\Omega}\mathbf{u}d\Omega+\int_{\Gamma}\mathbf{F}\cdot\mathbf{n}d\Gamma=0 \quad , \tag{5.1}$$

where

$$\mathbf{u}=\left\{\begin{array}{c}\rho\\ \rho v_x\\ \rho v_y\\ \rho v_z\\ \rho e\end{array}\right\},\ \mathbf{F}=\left\{\begin{array}{ccc}\rho v_x & \rho v_y & \rho v_z\\ \rho v_x^2+p & \rho v_x v_y & \rho v_x v_z\\ \rho v_y v_x & \rho v_y^2+p & \rho v_y v_z\\ \rho v_z v_x & \rho v_z v_y & \rho v_z^2+p\\ v_x\left(\rho e+p\right) & v_y\left(\rho e+p\right) & v_z\left(\rho e+p\right)\end{array}\right\}, \tag{5.2}$$

and

$$p=\left(\gamma-1\right)\rho\left[e-\frac{1}{2}\left\|\mathbf{v}\right\|^2\right]. \tag{5.3}$$

Here $\rho, v_x, v_y, v_z, e, p$ and $\gamma$ denote, respectively, the density, x,y,z velocities, total energy, pressure and ratio of ratio of specific heats. The equations are discretized using a cell-centered finite-volume scheme of the form:

$$\mathrm{vol}_i\frac{d\mathbf{u}_i}{dt}=\mathbf{R}_i=-\sum_{faces}\underline{\mathbf{F}}\cdot\underline{\mathbf{s}} \tag{5.4}$$

where

$$\underline{\mathbf{F}}\cdot\underline{\mathbf{s}}=\left\|\underline{\mathbf{s}}\right\|\left[f_i+f_j-\beta\cdot\lambda_{\max}\cdot\left(u_i-u_j\right)\right], \tag{5.5}$$

$$f_i = \frac{\underline{\mathbf{s}}}{\|\underline{\mathbf{s}}\|} \cdot \underline{\mathbf{F}} \quad , \quad \lambda_{\max} = \|\underline{\mathbf{v}}\| + c, \tag{5.6}$$

where $\text{vol}_i$ denotes the volume of the $i$th element, $\underline{\mathbf{s}}$ denotes the face normal, and $\beta$ is a parameter controlling the amount of artificial viscosity.

## 5.3 Implementation on Graphics Hardware

### 5.3.1 Overview

The performance-critical portion of the code is a loop which repeatedly computes the time derivatives of the conserved variables, (5.4), and updates them using an explicit time-stepping scheme. By far the most expensive computation performed is that of computing the residual or right-hand-side $\mathbf{R}_i$ in (5.4), and consists of accumulating flux contributions and artificial viscosity. Therefore, the performance of the CUDA kernel which implements this computation is crucial in determining whether or not high performance is achieved, and is the focus of this section. The code implementing this step is listed in Appendix B. It will be necessary to reference a number of technical details regarding modern graphics hardware, provided in the NVIDIA CUDA documentation [53].

### 5.3.2 Redundant Computation

The time derivative computation is parallelized on a per-element basis, with one thread per element. First, each thread reads in its volume, along with its conserved variables from *global memory* [53, Sec. 5.1.2.1], from which derived quantities such as the pressure, velocity, the speed of sound and the flux contribution are computed. The kernel then loops over each of the four faces of the tetrahedral element, in order to accumulate fluxes and artificial viscosity. The face's normal is read along with the index of the adjacent element, where this index is then used to access the adjacent element's conserved variables. The required derived quantities are computed and then the flux and artificial viscosity are accumulated

into the element's residual.

This approach requires redundant computation of flux contributions, and other quantities derived from the conserved variables. We could have instead chosen to precompute each element's flux contribution, so that each of its neighbors would not need to redundantly compute these. However, this approach turned out to be slower for two reasons. The first is that reading the flux contributions requires three times the amount of global memory access than just reading the conserved variables. The second is that the redundant computation can be performed simultaneously with global memory access, as described in [53, Sec. 5.1.2.1].

### 5.3.3 Numbering Scheme

The global memory access required for reading the conserved variables of neighboring elements is at risk of being highly *non-coalesced*, which would result in lower memory bandwidth [53, Sec. 3.1]. An implication of the coalescing requirements, for graphics hardware with compute capability 1.2 or higher [53, Page 54], is that for $i = 1, 2, 3, 4$, the $i$th neighbor of each consecutive element should be close as possible in memory. To achieve reasonably coalesced access, the elements are first numbered using the bin numbering scheme described by Löhner [42, Sec. 15.1.2.2]. This numbering works by overlaying a regular grid structure (of bins). Each point in the mesh is assigned to a bin. The points are then renumbered by assigning numbers while traversing the regular grid structure in a fixed fashion. In this preliminary work, the focus is not on the technical details of the particular numbering scheme used, but that the numbering scheme is using a reasonable heuristic in order to ensure that elements which are nearby in space are also nearby in memory. Then, the indices of the four neighbors of each tetrahedral element are sorted in increasing order to ensure that, for example, the second neighbor of consecutive elements are close in memory.

Several special cases have to be considered in order to deal with faces that are on the boundaries of the computational domain. In the present case, these are marked by storing a negative index in the connectivity array that refers to the particular boundary condition

desired (e.g. wing boundary, far-field, etc.). This results in possible branching, which incurs no significant penalty on modern graphics hardware, as long as all threads within a *warp* take the same branch [53, Page 14]. To minimize this penalty, in addition to having ensured that only the first face of each element can be a boundary face, we modify the bin-based ordering to ensure that boundary elements are stored consecutively in memory, which means that there can be at most two divergent warps.

### 5.3.4 Data-Dependent Memory Access and Shared Memory

*Shared memory* is an important feature of modern graphics hardware used to avoid redundant global memory access amongst threads within a block [53, Sec. 5.1.2]. The hardware does not automatically make use of shared memory, and it is therefore up to the software to explicitly specify how shared memory should be used. Thus information must be made available which specifies what global memory access can be shared by multiple threads within a block. For structured grid based solvers, this information is known a priori due to the fixed memory access pattern of such solvers. On the other hand, the memory access pattern of unstructured grid based solvers is data-dependent. With a per-element/thread based connectivity data structure, this information is not provided, and therefore shared memory is not applicable. It may be possible for unstructured grid based methods with a higher degree of connectivity to effectively use shared memory by collapsing a per-element/thread connectivity data structure into a per-thread-block connectivity data structure, and perform possibly redundant computation. However, for this to not be excessively wasteful would require a much higher degree of connectivity than the low-order, tetrahedral, unstructured grid based method considered here.

## 5.4 Results

The performance of the GPU code was measured on a prototype NVIDIA Tesla$^{TM}$ GPU, supporting compute capability 1.3, with 24 multiprocessors. The performance of the equivalent optimized OpenMP CPU code, compiled with the Intel C++ Compiler, version 10.1,

was measured with on an Intel Core 2 Quad CPU Q9450, running either one or four threads.

A NACA0012 wing in supersonic ($M_\infty = 1.2, \alpha = 0^o$) flow was used as a test case. The surface of the mesh, which has 1.6 million elements, is shown in Figure 5.1. The pressure contours are plotted in Figure 5.2. Timing measurements when running in single-precision are given in Figure 5.3 for a variety of meshes, showing an average performance scaling factor of 11.18x in comparison to the OpenMP code running on four cores and 39.38x in comparison to the OpenMP code on one core. Furthermore, the code running on graphics hardware is faster by a factor 3.86x using redundant computation in comparison to pre-computing flux contributions. Timing measurements when running in single-precision are given in Figure 5.4 for a variety of meshes, showing an average performance scaling factor of 1.87x in comparison to the OpenMP code running on four cores and 5.91x in comparison to the OpenMP code on one core. Furthermore, the code running on graphics hardware is faster by a factor 1.13x using redundant computation in comparison to pre-computing flux contributions.

A missile in supersonic ($M_\infty = 1.2, \alpha = 8^o$) flow was used as a test case. The pressure contours are plotted in Figure 5.5. Timing measurements when running in single-precision are given in Figure 5.7 for a variety of meshes, showing an average performance scaling factor of 12.53x in comparison to the OpenMP code running on four cores and 44.62x in comparison to the OpenMP code on one core. Furthermore, the code running on graphics hardware is faster by a factor 3.41x using redundant computation in comparison to pre-computing flux contributions. Timing measurements when running in single-precision are given in Figure 5.8 for a variety of meshes, showing an average performance scaling factor of 2.07x in comparison to the OpenMP code running on four cores and 6.63x in comparison to the OpenMP code on one core. Furthermore, the code running on graphics hardware is faster by a factor 1.13x using redundant computation in comparison to pre-computing flux contributions.

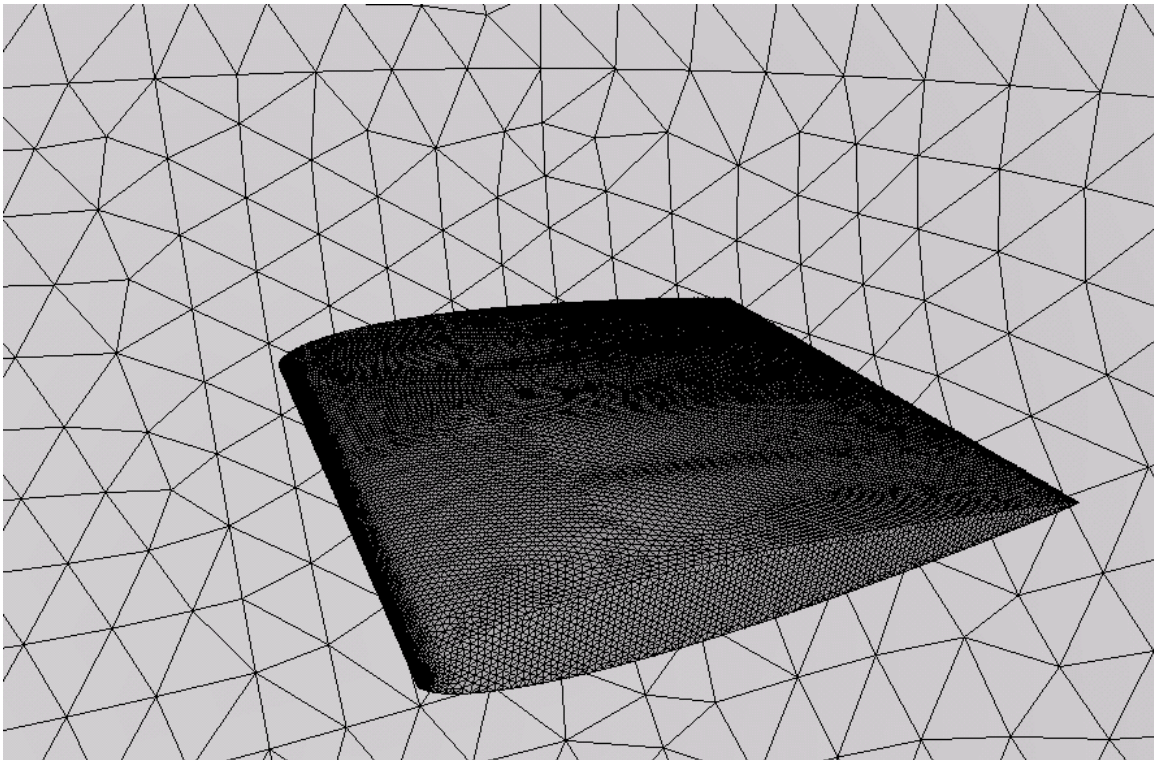


Figure 5.1: Surface Mesh Used (1.6 million elements)

## 5.5 Conclusions

A substantial performance gain has been achieved by using effective techniques which take advantage of the computational resources of modern graphics hardware. Based on these results, it is expected that current and future GPUs will be well-suited and widely used for unstructured grid based solvers. Such an order of magnitude speed-up can result in a significant increase in the scale and complexity of the problems considered in computational fluid dynamics.

While CUDA is proprietary, there has been a great push in industry for an open standard which has resulted in OpenCL. This standard is based upon CUDA, and therefore many of the techniques presented here will be of relevance moving forward.

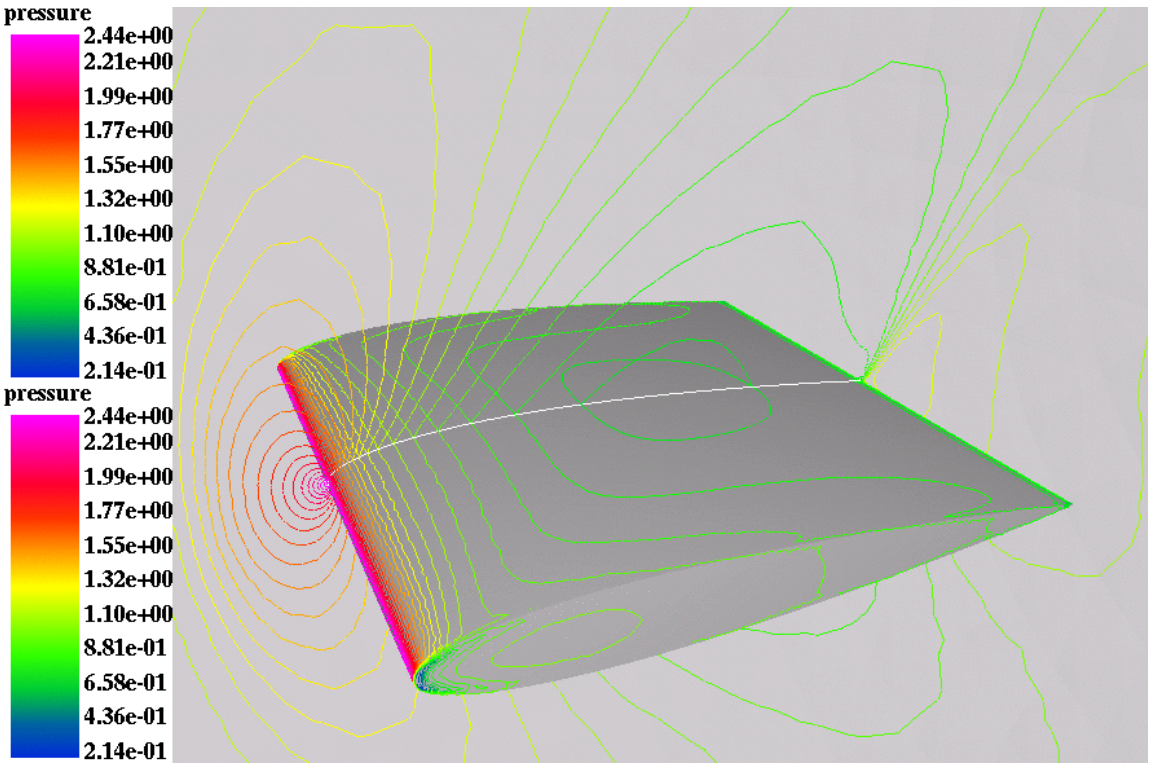



Figure 5.2: Pressures Obtained at the Surface and Plane for the NACA00012 Wing

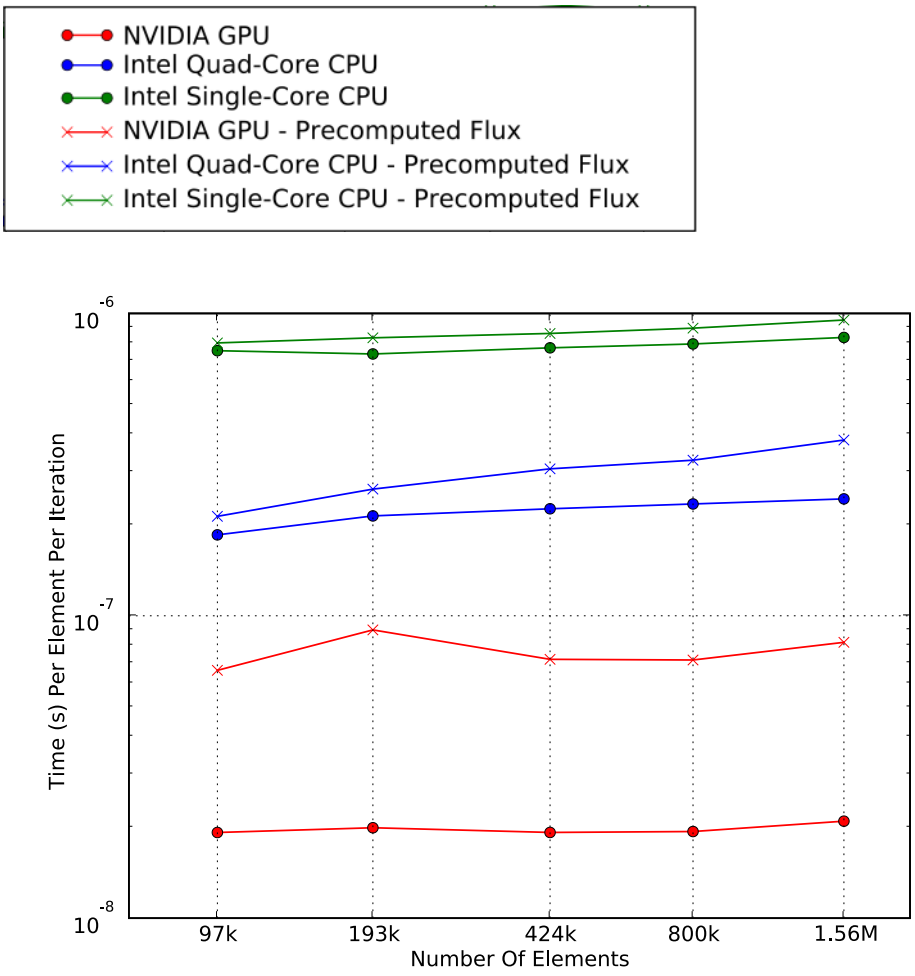



Figure 5.3: Running Time (s) Per Element Per Iteration for the NACA0012 Wing in Single-Precision.

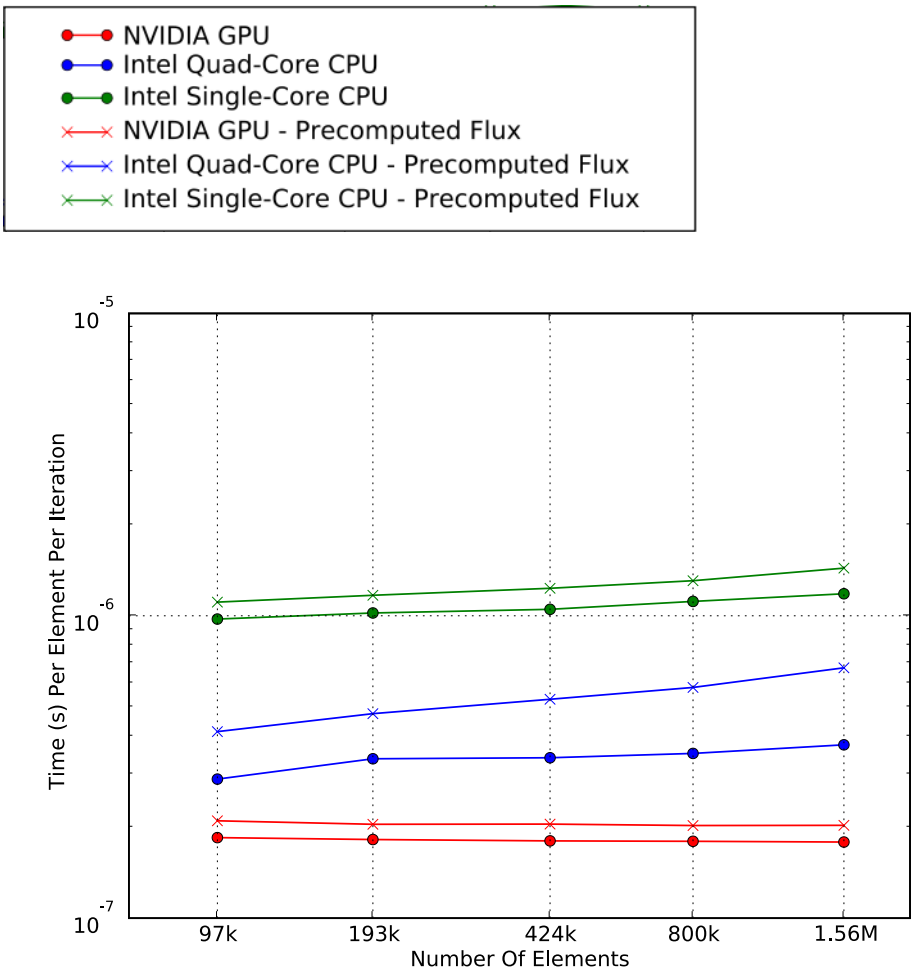



Figure 5.4: Running Time (s) Per Element Per Iteration for the NACA0012 Wing in Double-Precision.

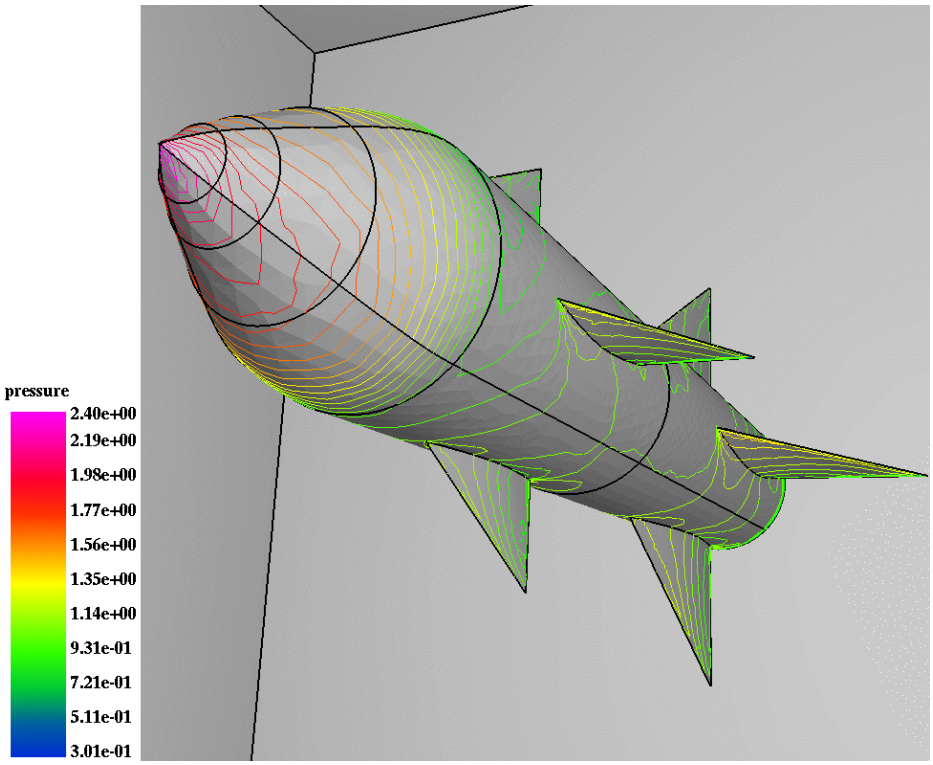



Figure 5.5: Pressures Obtained at the Surface for the Missile

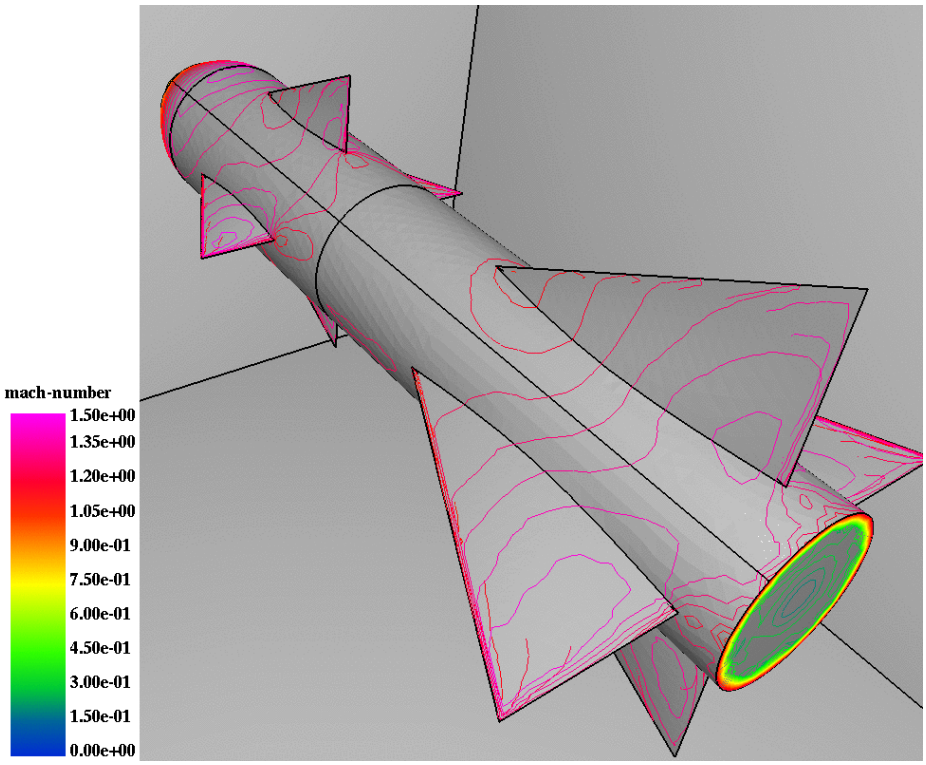



Figure 5.6: Mach Number Obtained at the Surface for the Missile

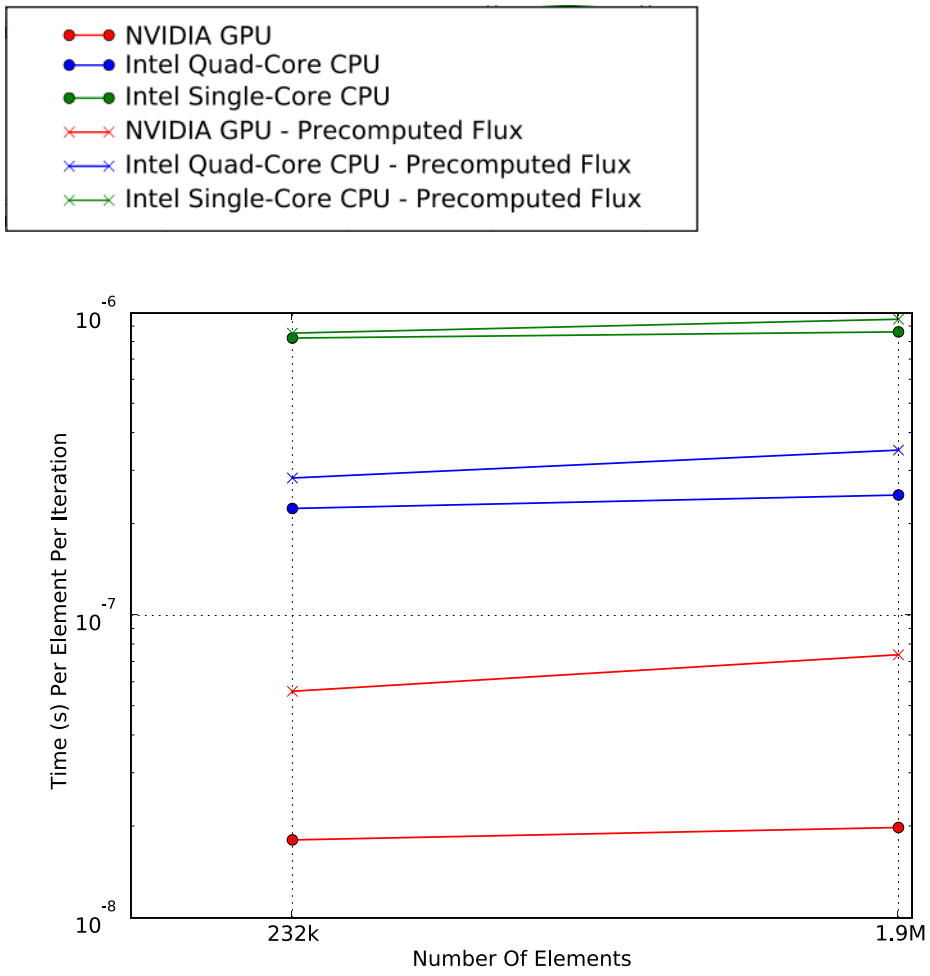



Figure 5.7: Running Time (s) Per Element Per Iteration for the Missile in Single-Precision.

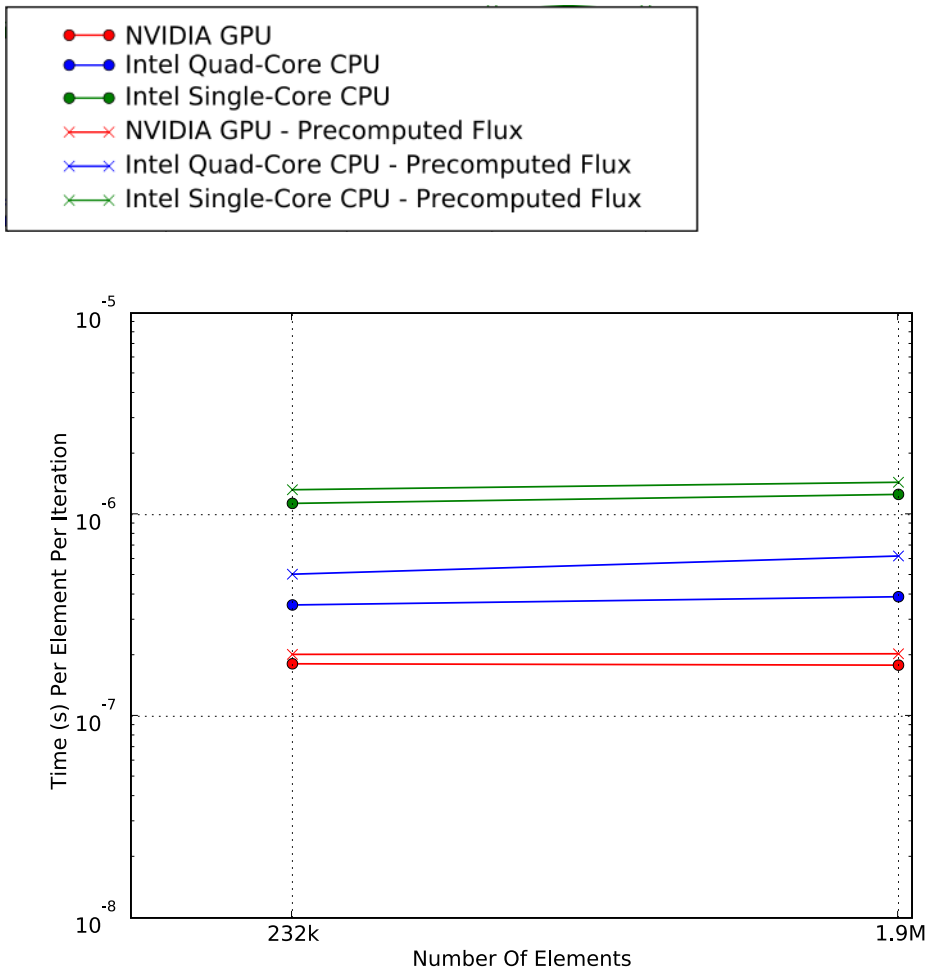



Figure 5.8: Running Time (s) Per Element Per Iteration for the Missile in Double-Precision.

# Chapter 6: Conclusions

In this dissertation we have equipped kernel-based meshless methods with a number of new tools and features.

Previously, the only known general technique for employing adaptively-scaled kernel-based trial functions was shown to be not well-posed. We have presented an alternative, general technique for interpolation using adaptively-scaled trial functions. This was done by constructing an adaptively-scaled kernel. The key to obtaining well-posedness was maintaining positive definiteness during the kernel's construction. In this case, the adaptivity is achieved by transforming the underlying domain, regardless of the placement of trial function centers, so that, for example, trial functions may be scaled non-uniformly within their support. This allows for kernel-based interpolation to better deal with less uniform point distributions by improving stability. The previously considered approach scales each trial function uniformly within its support, but fails to be well-posed in general.

Previous error bounds obtained for unsymmetric kernel-based meshless methods using Schaback's framework were suboptimal for problems involving fractional order Sobolev spaces, such as inhomogeneous boundary value problems. This was due to a lack of optimal sampling inequalities for bounding fractional order Sobolev norms. We have extended previous sampling inequalities to optimally bound such norms, thus improving the convergence results obtained within Schaback's framework. We have also generalized sampling inequalities to incorporate higher-order samples. Within Schaback's framework this appears to lead to higher order convergence results, which however are still lower order than those obtained using interpolation, and thus improvements are possible. The sampling inequality obtained in this dissertation is a general statement regarding functions in a Sobolev spaces and may have diverse applications.

We have provided kernel-based meshless methods with a direct visualization technique,

by adapting Fourier volume rendering to deal directly with meshless data, which was previously only used directly for grid-based data. Therefore, this technique does not require first sampling the meshless data. Furthermore, it considers the true form of the data: a function defined in three-dimensional space, not just a collection of samples at scattered points in space, as many previous meshless visualization techniques do. Because this adaptation is direct, it avoids the prohibitive computational and memory costs and quality loss of discretizing the meshless data into a three-dimensional grid. This technique is more well-suited for data with mostly low frequency content. Conversely, splatting is better suited for data with low spatial support, and thus the two can be seen as complementary. Therefore, a hybrid technique is possible which combines both techniques.

Finally, we implemented an unstructured grid-based inviscid, compressible flow solver on modern graphics hardware, an extremely powerful architecture for scientific computing. This architecture, however, has fundamentally different requirements for high-performance than those of more traditional CPU architectures. The main issue for this application was obtaining high effective memory bandwidth. We presented techniques for achieving coalesced memory access in the face of an unstructed, data-dependent memory access pattern, and obtained an order of magnitude speed-up in comparison to an equivalent code running on a quad-core CPU. Efficient implementations of kernel-based meshless methods, will also exhibit similar irregular and data-dependent memory access.

# Appendix A: Code Listing: Meshless Data Fourier Transform Sampling in CUDA

```
inline __device__ float fourier_transform_sph(float r)
{
        if(r >= 0.06f)
        {
                float m = CUDART_PI_F*r;
                float cos_2_m, sin_2_m;
                __sincosf(2.0f*m, &sin_2_m, &cos_2_m);
                return ((2.3561944901923448f)/(m*m*m*m*m*m))
                        *(cos_2_m-1.0f)*(cos_2_m+m*sin_2_m-1.0f);
        }
        // a quadratic interpolant, note that CUDART_PI_F is the value at zero
        return CUDART_PI_F + r*(-0.007968913156311f + r*(-18.293608272337678f)));
}

template <int block_length, bool is_first_group, bool has_radii>
__global__ void sample_fourier_transform_over_grid(Group group, VisConfig vis_config)
{
        // setup pointers to shared memory
        extern __shared__ float shared[];
        Constraint* ds_constraints = (Constraint*)shared;
        float* ds_radii;
        if(has_radii) ds_radii = (float*)(ds_constraints + block_length);

        int image_size = 2*vis_config._cutoff_frequency.x*vis_config._cutoff_frequency.y;
        int index = (block_length*blockIdx.x + threadIdx.x);
        int x = index % (2*vis_config._cutoff_frequency.x);
        if(x > vis_config._cutoff_frequency.x) x = x-(2*vis_config._cutoff_frequency.x);
        int y = (index % image_size) / (2*vis_config._cutoff_frequency.x);
        int partial_sum_index = index / image_size;
        int number_of_terms_per_partial_sum = group.d_number_of_terms
                                                / vis_config._number_of_partial_sums;
        int first_term = partial_sum_index*number_of_terms_per_partial_sum;
        int last_term = first_term + number_of_terms_per_partial_sum;

        // compute the image space coordinates
        float fu = vis_config.step_size.x*x, fv = vis_config.step_size.y*y;

        // map from image space into frequency space
        float3 f_coord = make_float3(fu*vis_config.u_axis.x + fv*vis_config.v_axis.x,
```

```
                        fu*vis_config.u_axis.y + fv*vis_config.v_axis.y,
                        fu*vis_config.u_axis.z + fv*vis_config.v_axis.z);

float r = sqrtf((float)(fu*fu + fv*fv));

// loop over the terms in the partial summation
float2 sum = make_float2(0.0f, 0.0f);
float sin_v, cos_v;
Constraint constraint;
float r0;
int thread_index = threadIdx.x;
for(unsigned int k = first_term; k < last_term; k += block_length)
{
        // step 1: stage global memory into shared memory.  this access is coalesced
        //          and should be minimal since there is one 128-bit read
        //          and one 32-bit read
        ds_constraints[thread_index] = group.d_constraints[k + thread_index];
        if(has_radii) ds_radii[thread_index] = group.d_radii[k + thread_index];
        __syncthreads();

        //step 2: for each point, accumulate the terms in shared memory
        //         there are no bank conflicts since the same data is broadcast
        //         to every thread
        for(unsigned int j = 0; j != block_length; j++)
        {
                constraint = ds_constraints[j];
                float term;
                r0 = ds_radii[j];
                term = fourier_transform_sph(r*r0);
                __sincosf(CUDART_2PI_F*dot(f_coord, constraint.position), &sin_v, &cos_v);
                sum.x += constraint.weight*term*cos_v;
                sum.y += constraint.weight*term*sin_v;
        }
        __syncthreads();
}

if(is_first_group)
{
        vis_config._d_freq_image[index]
                = make_float2(sum.x*vis_config._scale, -sum.y*vis_config._scale);
}
else
{
        float2 prev = vis_config._d_freq_image[index];
        vis_config._d_freq_image[index]
```

```
                    = make_float2(prev.x + sum.x*vis_config._scale,
                                  prev.y - sum.y*vis_config._scale);
        }
}
```

# Appendix B: Code Listing: Flux Computation in CUDA

```
__device__ __host__ inline void compute_flux_contribution(float& density, float3& momentum,
        float& density_energy, float& pressure, float3& velocity, float3& fc_momentum_x,
        float3& fc_momentum_y, float3& fc_momentum_z, float3& fc_density_energy)
{
        fc_momentum_x.x = velocity.x*momentum.x + pressure;
        fc_momentum_x.y = velocity.x*momentum.y;
        fc_momentum_x.z = velocity.x*momentum.z;



        fc_momentum_y.x = fc_momentum_x.y;
        fc_momentum_y.y = velocity.y*momentum.y + pressure;
        fc_momentum_y.z = velocity.y*momentum.z;

        fc_momentum_z.x = fc_momentum_x.z;
        fc_momentum_z.y = fc_momentum_y.z;
        fc_momentum_z.z = velocity.z*momentum.z + pressure;

        float de_p = density_energy+pressure;
        fc_density_energy.x = velocity.x*de_p;
        fc_density_energy.y = velocity.y*de_p;
        fc_density_energy.z = velocity.z*de_p;
}

__device__ inline void compute_velocity(float& density, float3& momentum, float3& velocity)
{
        velocity.x = momentum.x / density;
        velocity.y = momentum.y / density;
        velocity.z = momentum.z / density;
}

__device__ inline float compute_speed_sqd(float3& velocity)
{
        return velocity.x*velocity.x + velocity.y*velocity.y + velocity.z*velocity.z;
}

__device__ inline float compute_pressure(float& density, float& density_energy, float& speed_sqd)
{
        return (float(GAMMA)-float(1.0f))*(density_energy - float(0.5f)*density*speed_sqd);
}

__device__ inline float compute_speed_of_sound(float& density, float& pressure)
{
```

```
        return sqrtf(float(GAMMA)*pressure/density);
}

__global__ void cuda_compute_flux(int nelr, int* elements_surrounding_elements,
                                  float* normals, float* variables, float* fluxes)
{
        const float smoothing_coefficient = float(0.2f);
        const int i = (blockDim.x*blockIdx.x + threadIdx.x);

        int j, nb;
        float3 normal; float normal_len;
        float fct;

        float density_i = variables[i + VAR_DENSITY*nelr];
        float3 momentum_i;
        momentum_i.x = variables[i + (VAR_MOMENTUM+0)*nelr];
        momentum_i.y = variables[i + (VAR_MOMENTUM+1)*nelr];
        momentum_i.z = variables[i + (VAR_MOMENTUM+2)*nelr];

        float density_energy_i = variables[i + VAR_DENSITY_ENERGY*nelr];

        float3 velocity_i;      compute_velocity(density_i, momentum_i, velocity_i);
        float speed_sqd_i       = compute_speed_sqd(velocity_i);
        float speed_i           = sqrtf(speed_sqd_i);
        float pressure_i        = compute_pressure(density_i, density_energy_i, speed_sqd_i);
        float speed_of_sound_i = compute_speed_of_sound(density_i, pressure_i);
        float3 fc_i_momentum_x,
                fc_i_momentum_y,
                fc_i_momentum_z;
        float3 fc_i_density_energy;
        compute_flux_contribution(density_i, momentum_i,
                                  density_energy_i, pressure_i, velocity_i,
                                  fc_i_momentum_x, fc_i_momentum_y,
                                  fc_i_momentum_z, fc_i_density_energy);

        float flux_i_density = float(0.0f);
        float3 flux_i_momentum;
        flux_i_momentum.x = float(0.0f);
        flux_i_momentum.y = float(0.0f);
        flux_i_momentum.z = float(0.0f);
        float flux_i_density_energy = float(0.0f);

        float3 velocity_nb;
        float density_nb, density_energy_nb;
        float3 momentum_nb;
```

```
float3 fc_nb_momentum_x,
       fc_nb_momentum_y,
       fc_nb_momentum_z;
float3 fc_nb_density_energy;
float speed_sqd_nb, speed_of_sound_nb, pressure_nb;

#pragma unroll
for(j = 0; j < NNB; j++)
{
       nb = elements_surrounding_elements[i + j*nelr];
       normal.x = normals[i + (j + 0*NNB)*nelr];
       normal.y = normals[i + (j + 1*NNB)*nelr];
       normal.z = normals[i + (j + 2*NNB)*nelr];
       normal_len = sqrtf(normal.x*normal.x + normal.y*normal.y + normal.z*normal.z);

       if(nb >= 0)      // a regular neighbor
       {
              density_nb = variables[nb + VAR_DENSITY*nelr];
              momentum_nb.x = variables[nb + (VAR_MOMENTUM+0)*nelr];
              momentum_nb.y = variables[nb + (VAR_MOMENTUM+1)*nelr];
              momentum_nb.z = variables[nb + (VAR_MOMENTUM+2)*nelr];
              density_energy_nb = variables[nb + VAR_DENSITY_ENERGY*nelr];
                                  compute_velocity(density_nb, momentum_nb, velocity_nb);
              speed_sqd_nb      = compute_speed_sqd(velocity_nb);
              pressure_nb       = compute_pressure(density_nb, density_energy_nb, speed_sqd_nb);
              speed_of_sound_nb = compute_speed_of_sound(density_nb, pressure_nb);
              compute_flux_contribution(density_nb, momentum_nb, density_energy_nb,
                                        pressure_nb, velocity_nb,
                                        fc_nb_momentum_x, fc_nb_momentum_y, fc_nb_momentum_z,
                                        fc_nb_density_energy);

              // artificial viscosity
              fct = -normal_len*smoothing_coefficient*float(0.5f)
                     *(speed_i + sqrtf(speed_sqd_nb) + speed_of_sound_i + speed_of_sound_nb);
              flux_i_density += fct*(density_i-density_nb);
              flux_i_density_energy += fct*(density_energy_i-density_energy_nb);
              flux_i_momentum.x += fct*(momentum_i.x-momentum_nb.x);
              flux_i_momentum.y += fct*(momentum_i.y-momentum_nb.y);
              flux_i_momentum.z += fct*(momentum_i.z-momentum_nb.z);

              // accumulate cell-centered fluxes
              fct = float(0.5f)*normal.x;
              flux_i_density += fct*(momentum_nb.x+momentum_i.x);
              flux_i_density_energy += fct*(fc_nb_density_energy.x+fc_i_density_energy.x);
              flux_i_momentum.x += fct*(fc_nb_momentum_x.x+fc_i_momentum_x.x);
```

```
        flux_i_momentum.y += fct*(fc_nb_momentum_y.x+fc_i_momentum_y.x);
        flux_i_momentum.z += fct*(fc_nb_momentum_z.x+fc_i_momentum_z.x);

        fct = float(0.5f)*normal.y;
        flux_i_density += fct*(momentum_nb.y+momentum_i.y);
        flux_i_density_energy += fct*(fc_nb_density_energy.y+fc_i_density_energy.y);
        flux_i_momentum.x += fct*(fc_nb_momentum_x.y+fc_i_momentum_x.y);
        flux_i_momentum.y += fct*(fc_nb_momentum_y.y+fc_i_momentum_y.y);
        flux_i_momentum.z += fct*(fc_nb_momentum_z.y+fc_i_momentum_z.y);

        fct = float(0.5f)*normal.z;
        flux_i_density += fct*(momentum_nb.z+momentum_i.z);
        flux_i_density_energy += fct*(fc_nb_density_energy.z+fc_i_density_energy.z);
        flux_i_momentum.x += fct*(fc_nb_momentum_x.z+fc_i_momentum_x.z);
        flux_i_momentum.y += fct*(fc_nb_momentum_y.z+fc_i_momentum_y.z);
        flux_i_momentum.z += fct*(fc_nb_momentum_z.z+fc_i_momentum_z.z);
    }
    else if(nb == -1)        // a wing boundary
    {
        flux_i_momentum.x += normal.x*pressure_i;
        flux_i_momentum.y += normal.y*pressure_i;
        flux_i_momentum.z += normal.z*pressure_i;
    }
    else if(nb == -2) // a far field boundary
    {
        fct = float(0.5f)*normal.x;
        flux_i_density += fct*(ff_variable[VAR_MOMENTUM+0]+momentum_i.x);
        flux_i_density_energy += fct*(ff_fc_density_energy[0].x+fc_i_density_energy.x);
        flux_i_momentum.x += fct*(ff_fc_momentum_x[0].x + fc_i_momentum_x.x);
        flux_i_momentum.y += fct*(ff_fc_momentum_y[0].x + fc_i_momentum_y.x);
        flux_i_momentum.z += fct*(ff_fc_momentum_z[0].x + fc_i_momentum_z.x);

        fct = float(0.5f)*normal.y;
        flux_i_density += fct*(ff_variable[VAR_MOMENTUM+1]+momentum_i.y);
        flux_i_density_energy += fct*(ff_fc_density_energy[0].y+fc_i_density_energy.y);
        flux_i_momentum.x += fct*(ff_fc_momentum_x[0].y + fc_i_momentum_x.y);
        flux_i_momentum.y += fct*(ff_fc_momentum_y[0].y + fc_i_momentum_y.y);
        flux_i_momentum.z += fct*(ff_fc_momentum_z[0].y + fc_i_momentum_z.y);

        fct = float(0.5f)*normal.z;
        flux_i_density += fct*(ff_variable[VAR_MOMENTUM+2]+momentum_i.z);
        flux_i_density_energy += fct*(ff_fc_density_energy[0].z+fc_i_density_energy.z);
        flux_i_momentum.x += fct*(ff_fc_momentum_x[0].z + fc_i_momentum_x.z);
        flux_i_momentum.y += fct*(ff_fc_momentum_y[0].z + fc_i_momentum_y.z);
        flux_i_momentum.z += fct*(ff_fc_momentum_z[0].z + fc_i_momentum_z.z);
```

```
            }
        }

        fluxes[i + VAR_DENSITY*nelr] = flux_i_density;
        fluxes[i + (VAR_MOMENTUM+0)*nelr] = flux_i_momentum.x;
        fluxes[i + (VAR_MOMENTUM+1)*nelr] = flux_i_momentum.y;
        fluxes[i + (VAR_MOMENTUM+2)*nelr] = flux_i_momentum.z;
        fluxes[i + VAR_DENSITY_ENERGY*nelr] = flux_i_density_energy;
}
```

# Bibliography

# Bibliography


[1] A. Antunes and J. Wallin, *Convergence on N-body plus SPH*, Bulletin of the American Astronomical Society, December 2001, pp. 1433–+.

[2] Rémi Arcangéli, María Cruz López de Silanes, and Juan José Torrens, *An extension of a bound for functions in Sobolev spaces, with applications to (m,s)-spline interpolation and smoothing*, Numerische Mathematik **107** (2007), 181–211.

[3] Nachman Aronszajn, *Theory of reproducing kernels*, Transactions of the American Mathematical Society **68** (1950), 337–404.

[4] Ivo Babuška, Uday Banerjee, John E. Osborn, and Qiaolun Li, *Quadrature for meshless methods*, International Journal for Numerical Methods in Engineering **76** (2008), 1434–1470.

[5] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl, *Meshless methods: An overview and recent developments*, Computer Methods in Applied Mechanics and Engineering **139** (1996), 3–47.

[6] Jean Bourgain, Haim Brezis, and Petru Mironescu, *Optimal control and partial differential equations, in honour of Professor Alain Bensoussan's 60th birthday*, ch. Another Look at Sobolev Spaces, pp. 439–455, IOS Press, 2001.

[7] Mira Bozzini, Licia Lenarduzzi, and Robert Schaback, *Adaptive interpolation by scaled multiquadrics*, Adv. in Comp. Math **16** (2002), 375–387.

[8] T. Brandvik and G. Pullan, *Acceleration of a two-dimensional Euler flow solver using commodity graphics hardware*, J. Proc. of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science **221** (2007), 1745–1748.

[9] ———, *Acceleration of a 3D Euler solver using commodity graphics hardware*, 46th AIAA Aerospace Sciences Meeting and Exhibit, January 2008.

[10] Susanne C. Brenner and L. Ridgway Scott, *The mathematical theory of finite element methods*, third ed., Texts in Applied Mathematics, vol. 15, Springer, New York, 2008.

[11] Martin Buhmann, *Radial basis functions: Theory and implementations*, Cambridge University Press, 2003.

[12] R. Cunha C. Scheidegger, J. Comba, *Practical CFD simulations on the GPU using SMAC.*, Computer Graphics Forum **24** (2005), 715–728.

[13] Philippe G. Ciarlet, *The finite element method for elliptic problems*, Classics in Applied Mathematics, vol. 40, SIAM, Philadelphia, 2002.

[14] Christopher S. Co and Kenneth I. Joy, *Isosurface generation for large-scale scattered data visualization*, Proceedings of VMV 2005 (Guenther Greiner, Joachim Hornegger, Heinrich Niemann, and Marc Stamminger, eds.), 2005, pp. 233–240.

[15] A. Corrigan and H.Q. Dinh, *Computing and rendering implicit surfaces composed of radial basis functions on the GPU*, Poster proceedings of the International Workshop on Volume Graphics, June 2005.

[16] Andrew Corrigan, John Wallin, and Matej Vesenjak, *Progress on meshless methods*, Computational Methods in Applied Sciences, vol. 11, ch. Visualization of meshless simulations using Fourier volume rendering, pp. 291–305, Springer, 2008.

[17] Tobin A. Driscoll and Alfa R.H. Heryudono, *Adaptive residual subsampling methods for radial basis function interpolation and collocation problems*, Computers & Mathematics with Applications **53** (2007), no. 6, 927 – 939.

[18] Yong Duan, *A note on the meshless method using radial basis functions*, Computers & Mathematics with Applications **55** (2008), 66–75.

[19] J. Duchon, *Splines minimizing rotation-invariant semi-norms in sobolev spaces*, pp. 85–100, Springer, 1977.

[20] Shane Dunne, Sandy Napel, and Brian Rutt, *Fast reprojection of volume data*, Proceedings of the First Conference on Visualization in Biochemical Computing, 1990, pp. 11–18.

[21] Alireza Entezari, Randy Scoggins, Torsten Möller, and Raghu Machiraju, *Shading for Fourier volume rendering*, VVS '02: Proceedings of the 2002 IEEE symposium On Volume Visualization and Graphics, 2002, pp. 131–138.

[22] Lawrence Evans, *Partial differential equations*, AMS, 2002.

[23] G. Fasshauer, *Solving partial differential equations by collocation with radial basis functions*, Surface Fitting and Multiresolution Methods, Vanderbilt University Press, 1997, pp. 131–138.

[24] Gregory E. Fasshauer, *Meshfree approximation methods with matlab*, World Scientific, 2007.

[25] Bengt Fornberg and Julia Zuev, *The runge phenomenon and spatially variable shape parameters in rbf interpolation*, Computers & Mathematics with Applications **54** (2007), no. 3, 379 – 398.

[26] C. Franke and R. Schaback, *Solving partial differential equations by collocation using radial basis functions*, Applied Mathematics and Computation **93** (1998), no. 1, 73–82.

[27] R. Franke, *Scattered data interpolation: tests of some methods*, Math. Comp. **48** (1982), 181–200.

[28] Peter Giesl and Holger Wendland, *Meshless collocation: Error estimates with application to dynamical systems*, Preprint Göttingen/München 2006, to appear in SIAM Journal on Numerical Analysis, 2006.

[29] Rafael Gonzalez and Richard Woods, *Digital image processing*, second ed., Prentice-Hall, 2002.

[30] Nolan Goodnight, *CUDA/OpenGL fluid simulation*, NVIDIA Corporation, 2007.

[31] M. H. Gross, L. Lippert, R. Dittrich, and S. Häring, *Two methods for wavelet-based volume rendering*, Computers and Graphics **21** (1997), no. 2, 237–252.

[32] T.R. Hagen, K.-A. Lie, and J.R. Natvig, *Solving the euler equations on graphics processing units*, vol. 3994, pp. 220–227, Springer, 2006.

[33] Rolland L. Hardy, *Multiquadric equations of topography and other irregular surfaces*, Journal of Geophysical Research **76** (1971), 1905–1915.

[34] M.J. Harris, *Fast fluid dynamics simulation on the GPU*, pp. 637–665, Addison-Wesley, 2004.

[35] J. Hart, *Ray tracing implicit surfaces*, Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces, 1993, pp. 1–16.

[36] Y. C. Hon and Robert Schaback, *On unsymmetric collocation by radial basis functions*, Appl. Math. Comput. **119** (2001), no. 2-3, 177–186. MR MR1823674

[37] M. Hopf and T. Ertl, *Hierarchical splatting of scattered data*, Proc. IEEE Visualization, 2003, pp. 433–440.

[38] Y. Jang, M. Weiler, M. Hopf, J. Huang, D. Ebert, K. Gaither, and T. Ertl, *Interactively visualizing procedurally encoded scalar fields*, Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '04 (O. Deussen, C. Hansen, D. Keim, and D. Saupe, eds.), 2004.

[39] E.J. Kansa, *Multiquadrics - A scattered data approximation scheme with applications to computational fluid dynamics*, Comput. Math. App. **19** (1990), 147–161.

[40] P. Lancaster and K. Salkauskas, *Surfaces generated by moving least squares methods*, Mathematics of Computation **37** (1981), no. 155, 141–158.

[41] Marc Levoy, *Volume rendering using the Fourier projection-slice theorem*, Proceedings of the Conference on Graphics Interface '92 (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1992, pp. 61–69.

[42] Rainald Löhner, *Applied CFD techniques: An introduction based on finite element methods*, second ed., Wiley, 2008.

[43] Rainald Löhner and Eugenio Oñate, *An advancing front point generation technique*, Communications in Numerical Methods in Engineering **14** (1998), 1097–1108.

[44] W.R. Madych, *An estimate for multivariate interpolation ii*, Journal of approximation theory **142** (2006), 116–128.

[45] Tom Malzbender, *Fourier volume rendering*, ACM Trans. Graph. **12** (1993), no. 3, 233–250.

[46] Vladimir Maz'ya and Gunther Schmidt, *Approximate approximations*, AMS, 2007.

[47] J. Meredith and Kwan-Liu Ma, *Multiresolution view-dependent splat based volume rendering of large irregular data*, Proceedings of IEEE Symposium on Parallel and Large Data Visualization and Graphics, 2001.

[48] C.A. Micchelli, *Interpolation of scattered data: distance matrices and conditionally positive definite functions*, Constr. Approx. **2** (1986), 11–22.

[49] J J Monaghan, *Smoothed particle hydrodynamics*, Reports on Progress in Physics **68** (2005), no. 8, 1703–1759.

[50] Francis J. Narcowich, Joseph D. Ward, and Holger Wendland, *Sobolev bounds on functions with scattered zeros, with applications to radial basis function surface fitting*, Math. Comp. **74** (2005), no. 250, 743–763 (electronic).

[51] Francis J. Narcowich, Joseph D. Ward, and Holger Wendland, *Sobolev error estimates and a Bernstein inequality for scattered data interpolation via radial basis functions*, Constructive Approximation **24** (2006), 175–186.

[52] N. Neophytou, K. Mueller, K. T. McDonnell, W. Hong, X. Guan, H. Qin, and A. Kaufman, *GPU-accelerated volume splatting with elliptical RBFs*, Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization 2006, May 2006.

[53] NVIDIA Corporation, *NVIDIA CUDA 2.0 programming guide*, 2008.

[54] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krger, Aaron E. Lefohn, and Timothy J. Purcell, *A survey of general-purpose computation on graphics hardware*, Computer Graphics Forum **26** (2007), no. 1, 80–113.

[55] D.J. Price, *SPLASH: An interactive visualisation tool for SPH data*, Accepted to the Publications of the Astronomical Society of Australia, 2007.

[56] R. Rau and W. Strasser, *Direct volume rendering of irregular samples*, Visualization in Scientific Computing '95, 1995, pp. 72–80.

[57] Z. Ren, M. Vesenjak, and A. Öchsner, *Behaviour of cellular structures under impact loading: A computational study*, Materials Science Forum **566** (2008), 53–60.

[58] Christian Rieger, *Sampling inequalities and applications*, Ph.D. thesis, Göttingen, 2008.

[59] Christian Rieger and Barbara Zwicknagl, *Sampling inequalities for infinitely smooth functions, with applications to interpolation and machine learning*, To appear in Advances in Computational Mathematics, 2008.

[60] R. Schaback, *Creating surfaces from scattered data using radial basis functions*, Mathematical methods for curves and surfaces (Ulvik, 1994), Vanderbilt Univ. Press, Nashville, TN, 1995, pp. 477–496. MR MR1356989 (96g:65025)

[61] R. Schaback and H. Wendland, *Inverse and saturation theorems for radial basis function interpolation*, Math. Comp. **71** (2002), no. 238, 669–681 (electronic). MR MR1885620 (2003a:41018)

[62] ———, *Kernel techniques: from machine learning to meshless methods*, Acta Numer. **15** (2006), 543–639.

[63] Robert Schaback, *Unsymmetric meshless methods for operator equations*, Preprint Göttingen, 2006.

[64] ———, *Convergence of unsymmetric kernel-based meshless collocation methods*, SIAM Journal on Numerical Analysis **45** (2007), no. 1, 333–351.

[65] ———, *Recovery of functions from weak data using unsymmetric meshless kernel-based methods*, To appear in Applied Numerical Mathematics, 2007.

[66] ———, personal communication, April 2008.

[67] Paul Stark, *Fourier volume rendering of irregular data sets*, Master's thesis, Simon Fraser University, 2002.

[68] Jonas Tölke, *Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA*, Computing and Visualization in Science (2008).

[69] Takashi Totsuka and Marc Levoy, *Frequency domain volume rendering*, SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, 1993, pp. 271–278.

[70] M. Vesenjak, A. Öchsner, M. Hriberŝek, and Z. Ren, *Behaviour of cellular structures with fluid fillers under impact loading*, International Journal of Multiphysics **1** (2007), 101–122.

[71] M. Vesenjak, Z. Ren, H. Mulerschon, and S. Matthaei, *Computational modelling of fuel motion and its interaction with the reservoir structure*, Journal of Mechanical Engineering **52** (2006), 85–100.

[72] Manfred Weiler, Ralf Botchen, Simon Stegmaier, Thomas Ertl, Jingshu Huang, Yun Jang, David S. Ebert, and Kelly P. Gaither, *Hardware-assisted feature analysis and visualization of procedurally encoded multifield volumetric data*, IEEE Comput. Graph. Appl. **25** (2005), no. 5, 72–81.

[73] H. Wendland and C. Rieger, *Approximate interpolation with applications to selecting smoothing parameters*, Numerische Mathematik **101** (2005), 729–748.

[74] Holger Wendland, *Ein beitrag zur interpolation mit radialen basisfunktionen*, Ph.D. thesis, Göttingen, 1995.

[75] ———, *Scattered data approximation*, Cambridge University Press, 2005.

[76] J. Wertz, E.J. Kansa, and L. Ling, *The role of the multiquadric shape parameters in solving elliptic partial differential equations*, Computers & Mathematics with Applications **51** (2006), no. 8, 1335 – 1348, Radial Basis Functions and Related Multivariate Meshfree Approximation Methods: Theory and Applications.

# Curriculum Vitae

Andrew Corrigan was born in New York City, and grew up on Long Island in Merrick, New York. He graduated from John F. Kennedy High School in Bellmore, New York in June 2002. In May 2005, he completed his undergraduate studies in Computational Science at Stevens Institute of Technology in Hoboken, New Jersey, graduating as valedictorian. While an undergraduate student, he performed research under the supervision of Prof. H. Quynh Dinh in the implementation of the conjugate gradients method on graphics hardware and interactive visualization of implicit surfaces defined by radial basis function interpolants. In August 2006, he enrolled in a Master's program, while still under the supervision of Prof. Dinh, performing research on the adaptation of Fourier volume rendering to radial basis function interpolants. He completed two internships at Siemens Corporate Research in Princeton, New Jersey during the summers of 2005 and 2006, working on a project involving the geometric modeling of hearing aids, under the supervision of Dr. Greg Slabaugh. He enrolled in the Computational Sciences and Informatics Ph.D. program in the Department of Computational and Data Sciences at George Mason University in August 2006. He began research on kernel-based meshless methods under the supervision of Prof. John Wallin and Prof. Thomas Wanner. In addition, he performed research under the supervision of Prof. Wanner beginning in May 2007 in the application of computational homology to the study of pattern formation. During the summer of 2008, he was a research assistant for Prof. Rainald Löhner performing research in meshless methods and the implementation of unstructured grid solvers on graphics hardware.