SAL 2002- 3

The Architecture Of SAGE – A Meshfree System Based On RFM

I. Tsukanov, V. Shapiro

The Architecture Of SAGE – A Meshfree System Based On RFM

I.Tsukanov V.Shapiro[*]
Spatial Automation Laboratory
University of Wisconsin-Madison
1513 University Avenue
Madison, WI, 53706
USA

October 8, 2002

Abstract

In a meshfree system, a geometric model of a domain neither conforms to, nor is restricted by a spatial discretization. Such systems for engineering analysis offer numerous advantages over the systems that are based on traditional mesh-based methods, but they also requires radical approaches to enforcing boundary conditions and novel computational tools for differentiation, integration, and visualization of fields and solutions.

We show that all of these challenges can be overcome, and describe SAGE (Semi-Analytic Geometry Engine) – a successful system specifically intended for meshfree engineering analysis. Our approach and individual modules are based on Rvachev's Function Method (RFM) but the described techniques, algorithms, and software are applicable to all mesh-based and meshfree methods and have broad use beyond solutions of boundary value problems.

Keywords: Engineering analysis, meshfree method, R-functions, computer aided engineering

1 Introduction

Conventional methods of engineering analysis rely on various spatial discretizations (meshes, grids, etc.) that have to conform to the shape of the geometric object. The spatial elements in the disretization (nodes, segments, cells) are used to satisfy the prescribed boundary conditions, to construct basis functions with good approximation properties, to perform numerical integration, and to visualize the solution results. However, most geometric and solid models of engineering artifacts use boundary representations, constructive solid geometry, and other high-level representations of geometric domains [29]. This implies that the required spatial dicretizations must be obtained by a theoretically difficult and computationally expensive procedure known as *meshing.* Recent advances in meshing technology allows fully automatic meshing for many simple engineering problems and geometries, but the problem remains challenging for more general two- and three-dimensional problems where meshing now dominates both manual and computer solution time. Once constructed, meshes severely constrain the original geometric model, limiting possible changes, motions, and deformations such as those needed for shape optimization and dynamic simulations [31].

These difficulties with conforming spatial discretizations are largely responsible for the rise of meshless and meshfree methods that discretize not the geometric domain but the underlying functional space. A number of techniques with basis functions that do not have to conform to the geometry of the domain have been developed: smooth particle hydrodynamics (SPH) [13, 18], the diffuse element method (DEM) [15], the reproducing kernel particle method (RKPM) [12, 4], the HP cloud method [5], the meshless local Petrov-Galerkin (MLPG) approach [1], the partition of unity methods (PUM) [14], and others. For computational purposes, many of the meshfree methods in fact use "background" spatial grids that do not constrain to the shape of the geometric domain as it is shown in Figure 1. The geometric domain neither conforms to, nor is constrained by, such a background mesh, allowing arbitrary motions, changes, and deformations of the geometric domain. The price of this mesh-freedom is the increased computational cost (for example, in integration over the unmeshed geometric domain) and difficulty in satisfying the prescribed boundary conditions in the absence of the discretized boundary.

[*]Corresponding author. E-mail addresses: vshapiro@engr.wisc.edu, igor@sal-cnc.me.wisc.edu

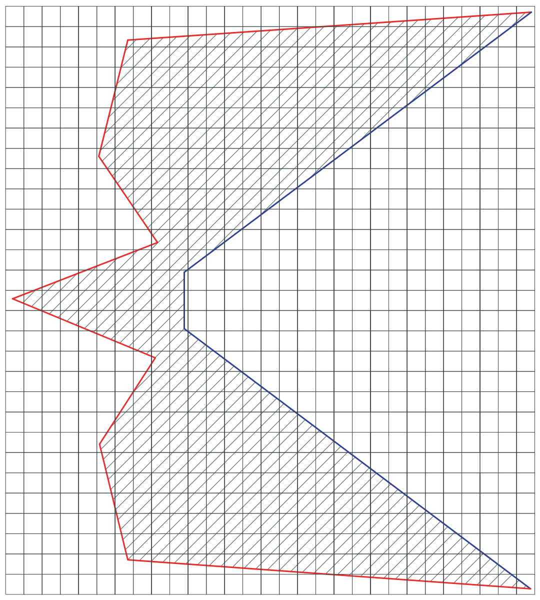


Figure 1: Meshfree methods use spatial discretization which may not conform to the shape of the geometric domain

In this paper we discuss the architecture of a meshfree system whose salient feature is the *exact treatment of all given boundary conditions.* This system implements the Rvachev's Function Method (RFM) [21, 23], based on the idea originally proposed by Kantorovich and further developed by Rvachev and his students in Ukraine. (RFM could also stand for the R-function method because it is effectively implemented with R-functions, but the ideas behind the method are substantially more general.)

Figure 2 shows the architecture of a typical engineering analysis system, which uses a geometric model, boundary conditions and governing equations of the problem as the input data and transforms this information into an approximate solution of the problem. Traditional PDE solvers require the original geometric model to be discretized (meshed) and the boundary conditioned to be applied at the nodes of the discretization. The architecture of the meshfree solver based on RFM is shown in Figure 3. The key principle of RFM requires an ability to construct functions that behave approximately as distances to the boundaries of the object where boundary conditions are prescribed. Naturally, this implies that such functions vanish on the corresponding boundaries. Such functions exist for all engineering artifacts, and Shapiro showed that such functions could be constructed automatically for a broad class of geometric domains [28]. We will describe briefly how to construct such functions automatically using theory of R-functions [30] in section 2.2. As diagrammed in Figure 3, these approximate distance functions, boundary conditions, and basis functions are assembled into a *solution structure* — a data structure that represents a space of all admissible solutions to the problem in hand. The solution structure satisfies the prescribed boundary conditions exactly and contains the necessary degrees of freedom in the form of coefficients of the basis functions. The linear combination of the basis functions can be used to approximate the differential equation of the problem. Any sufficiently complete system of basis functions can be used: polynomials, B-splines, functions forming a partition of unity, or even finite element functions.

This paper describes the data structures and algorithms needed for implementation of a meshfree system based on RFM. These include techniques for automatic construction of the functions satisfying the prescribed boundary conditions, automatic differentiation of such functions, adaptive integration and visualization over (non-meshed) geometrical models. We emphasize the issues related to automation and interaction of different subsystems. Theoretical grounds for the methods have been summarized in [23]. One such system, SAGE (Semi-Analytic Geometry Engine) has been fully implemented by the authors at the University of Wisconsin; a fully functional two-dimensional solver can be downloaded from http://sal-cnc.me.wisc.edu. RFM can also be viewed as a generalized finite element method with elements satisfying the given boundary conditions exactly. The RFM solution structure can also contain enrichment functions — functions that capture singularities and a priori known behaviour of the solution. And in fact, RFM may be combined with almost any other solution method, transforming it into another meshfree technique. In this

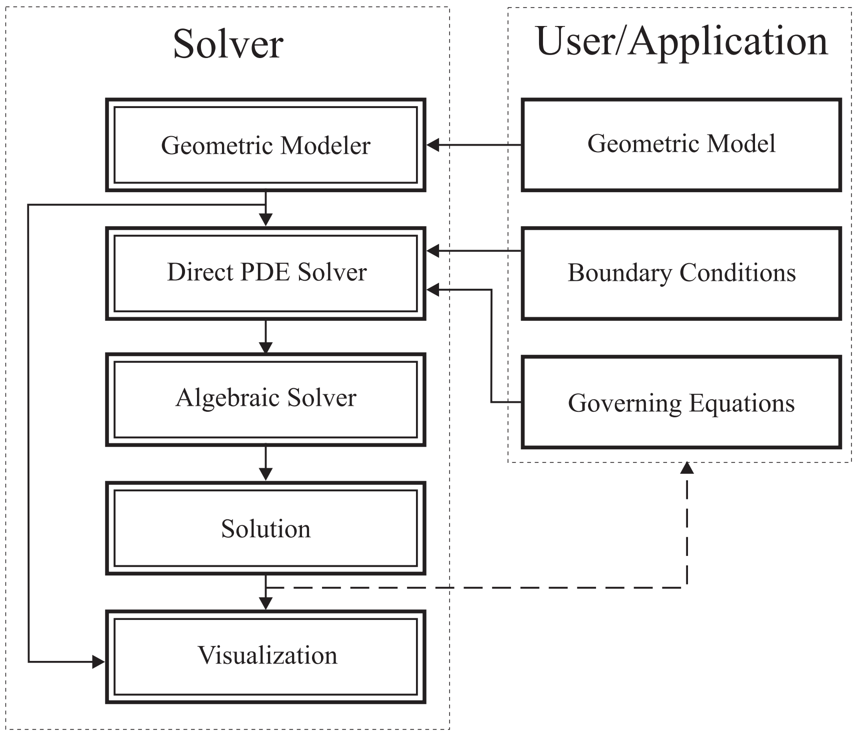


Figure 2: Architecture of a typical engineering analysis system

sense, the architecture of the system, as well as its components, have a broad appeal and wide applicability.

Previous implementations of RFM in systems POLYE have been reported by Ukrainian researchers [25], but these systems were aimed at mathematically sophisticated users who had to formulate the problem and program its solution manually in terms of functions. None of the previous systems attempted integration with geometric modelers or sought complete automation, as we do in this paper.

The rest of the paper is organized as follows: Section 2 explains the implementation of the automatic construction of the solution structures; Section 3 discusses the computational tools of the proposed meshfree system: automatic differentiation, adaptive integration and visualization. Section 4 illustrates application of the SAGE and the RFM to a variety of engineering problems.

2 Solution Structures Satisfying Boundary Conditions Exactly

2.1 Automatic Construction of RFM Solution Structures

Kantorovich proposed that a solution u to a boundary value problem with homogeneous Dirichlet boundary conditions can be constructed in the form of $u = \omega\Phi$, where $\omega = 0$ on the boundary and is positive elsewhere and $\Phi = \sum_{i=1}^{N} C_i\chi_i$ is a linear combination of basis functions χ_i with unknown coefficients C_i [10]. Various techniques can be employed to solve for C_i, for example, by minimization of an appropriate energy functional. The expression $\omega\Phi$ is the simplest example of a *solution structure* which represents a space of admissible solutions for the given problem with homogeneous Dirichlet boundary conditions.

The concept of solution structure is due to Rvachev [21, 23, 24] who observed that the term $u = \omega\Phi$ can be considered a remainder term in a special case of a generalized Taylor series expansion of u in the neighborhood of the boundary by the powers of the distance ω to the boundary. A straightforward generalization of this observation allows systematic construction of solution structures for any and all boundary conditions. In each case, the solution structure will exactly interpolate all values and derivatives prescribed on the boundary and will contain necessary degrees of

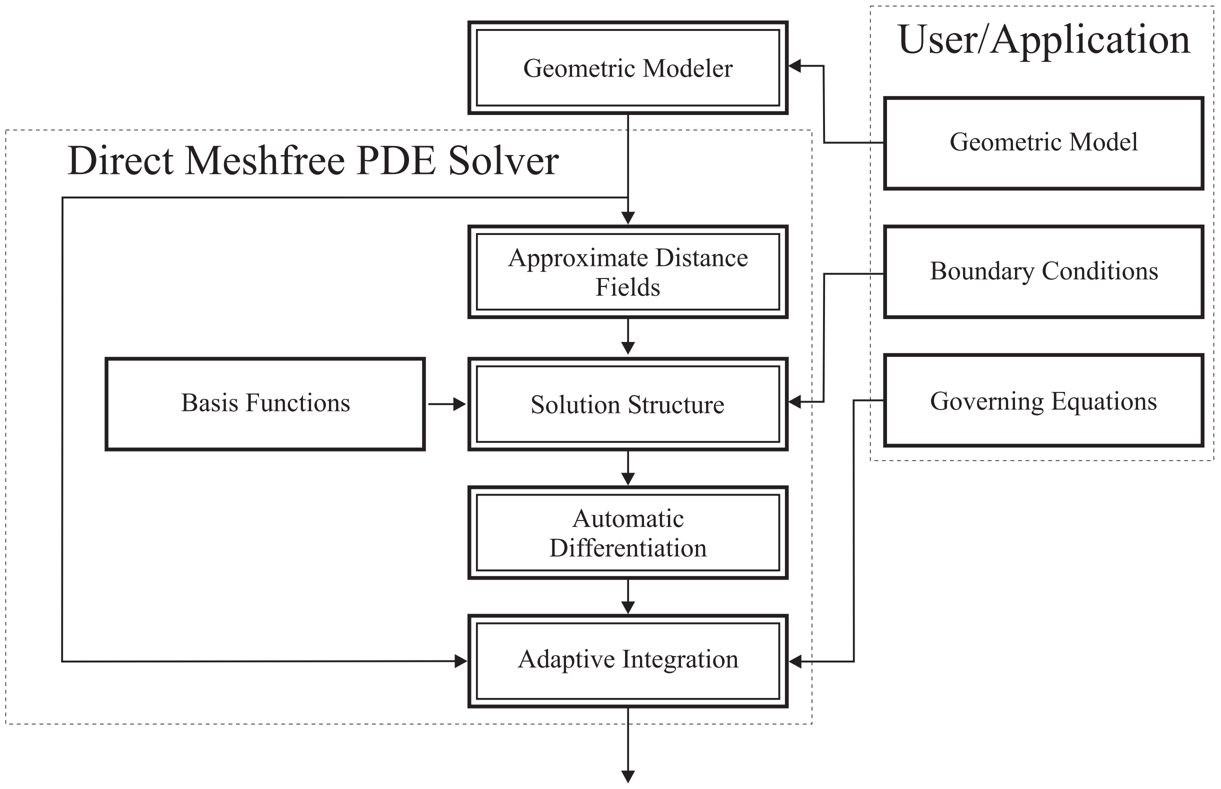


Figure 3: Implementation of the "Direct PDE solver" block shown in Figure 1 adapted for the RFM meshfree engineering analysis

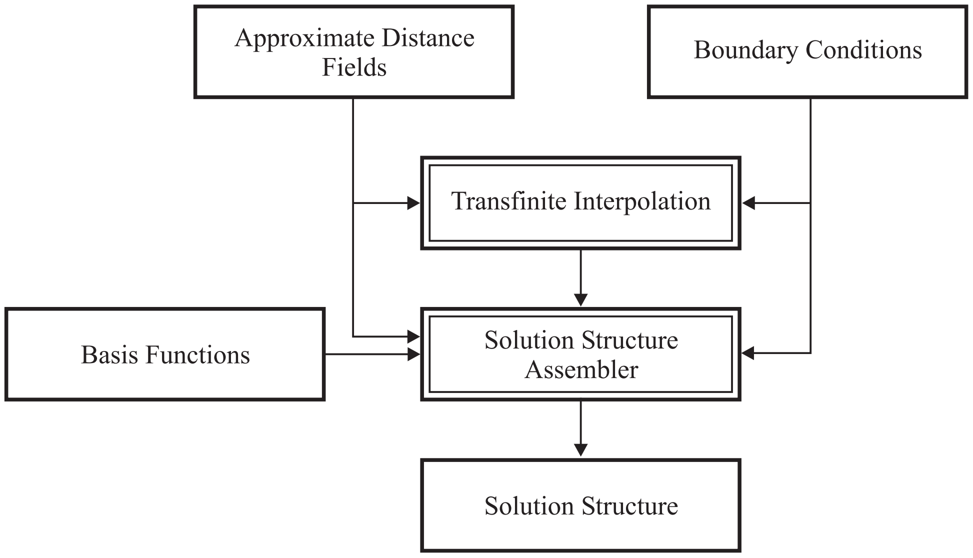

Approximate Distance Fields
Boundary Conditions
Transfinite Interpolation
Basis Functions
Solution Structure Assembler
Solution Structure

Figure 4: RFM solution structure comprises approximate distance fields, boundary conditions and basis functions

Type of Boundary Condition	Mathematical Formulation	Corresponding Solution Structure
Dirichlet	$u_{\|\partial\Omega} = \varphi$	$u = \omega\Phi + \varphi$
Neumann	$\frac{\partial u}{\partial n}_{\|\partial\Omega} = \varphi$	$u = \Phi - \omega D_1^\omega(\Phi) + \omega\varphi + \omega^2\Phi$
3-rd kind	$(\frac{\partial u}{\partial n} + hu)_{\|\partial\Omega} = \varphi$	$u = \Phi - \omega D_1^\omega(\Phi) - h\omega\Phi + \omega\varphi + \omega^2\Phi$
Mixed	$u_{\|\partial\Omega_1} = \varphi$ $(\frac{\partial u}{\partial n} + hu)_{\|\partial\Omega_2} = \psi$	$u = \omega_1\Phi + \frac{\omega_1\omega_2}{\omega_1+\omega_2}(\psi + \omega_2\Phi - D_1^{\omega_2}(\omega_1\Phi) - D_1^{\omega_2}(\varphi) - h\omega_1\Phi - h\varphi) + \varphi$

Table 1: Example solution structures corresponding to boundary conditions for the second order partial differential equation

freedom (approximating the remainder term in the Taylor series expansion) in order to approximate the governing equations of the problem. It can be shown that such a solution structure forms a complete space of functions that satisfy the given boundary conditions exactly and approximate the governing equations of the problem [23].

Solution structures have been derived and catalogued for most boundary conditions [21]. For example, Table 1 presents solution structures for most popular boundary conditions for second order partial differential equation. Function Φ in the solution structure represents a linear combination of basis functions $\{\chi_i\}_{i=1}^N$ with unknown coefficients C_i: $\Phi = \sum_{i=1}^N C_i\chi_i$. Different sets of coefficients result in different functions, but all of them satisfy the prescribed boundary conditions exactly — regardless of the chosen system of basis functions. Different combinations of boundary conditions are reduced to one of these solution structures. For instance, the solution structure for mixed boundary conditions is used for Dirichlet and Neumann boundary conditions. When the Neumann and 3-rd kind boundary conditions are prescribed the solution structure for 3-rd kind boundary conditions is employed. And so on.

These examples suggest a simple look-up system for a fully automatic construction process of RFM solution structures. The diagram in Figure 4 shows all ingredients required for construction of an RFM solution structure. Function ω in the solution structure is constructed as an approximate distance field using the theory of R-functions, as we explain in section 2.2. Depending on desired computational properties, basis functions can be selected from B-splines, polynomials, trigonomentric polynomials, or other popular choices. The original boundary conditions are accepted as arbitrary functions of spatial variables in the global coordinate system. Individual boundary conditions and/or solution structures are interpolated *transfinitely* [24], without discretization of approximation; the mathematical procedure is summarized in the Appendix and the corresponding computational procedure is discussed in section 2.3. The solution structure assembler constructs the actual solution structure after analyzing the prescribed boundary conditions, determining suitable combination with basis functions, and calling for construction of suitable distance fields.

More generally, the purpose of the solution structure is to incorporate into the computational procedure all a priori known information about solution [21, 23]. This information may include boundary conditions, asymptotic behavior and known singularities. Each solution structure consists of an approximation and an interpolation part. The interpolation part includes terms that interpolate the given functions from the boundaries and satisfy the non-homogeneous boundary conditions. The approximation part consists of terms of a solution structure that are represented by linear combinations of the basis functions with unknown coefficients. For a given boundary value problem and chosen basis functions, the approximate solution is obtained using variational, projection, or a variety of other numerical methods to solve for the numerical values of the coefficients.

Solution structures contain no information about the governing equations of the boundary value problems. The same solution structure can be used to represent solutions of different equations, but satisfying the same boundary conditions. The solution structure does not place any constraints on the choice of basis functions. In particular, this choice does not depend on any particular spatial discretization of the geometric domain or its boundary.

2.2 Approximate Distance Fields

We saw above that construction of a meshfree solution structure depends critically on the ability to automatically construct a distance field ω for (a portion of) the boundary of the geometric domain. Distance fields are widely used in computer graphics, robot motion planning, and other applications. Unfortunately, the distance functions are not differentiable at points equidistant from the boundary [3]. Moreover, the construction of the distance fields usually involves difficult and expensive geometric and numerical computations.

The limitations of the exact distance fields may be overcome by replacing them with smooth approximations, for example those that are constructed using the theory of R-functions. To be more precise, we propose to replace a distance field with its m-th order approximation in the following sense. Suppose point p is a *regular* point on the boundary of set S, i.e., a point where the unit normal direction ν is well defined. By definition, the exact distance function $f(p) = 0$; the first partial derivative of f at p in the direction of ν is equal to 1 and *all* higher derivatives vanish. This corresponds to the intuitive fact that the value of f at a point that is δ away from p is equal to δ. A suitable m-th order approximation of f is a function ω that is obtained by requiring that only *some* of the higher order derivatives vanish, that is for all regular points p on the boundary of S:

$$\frac{\partial \omega}{\partial \bar{\nu}} = 1; \frac{\partial^k \omega}{\partial \bar{\nu}^k} = 0; k = 2, 3...., m \tag{1}$$

Such a function ω is said to be normalized to the m-th order. Normalized functions behave like a distance function near their zero set (corresponding to the boundary of set S) and tend to smoothly approximate the distance function away from S.

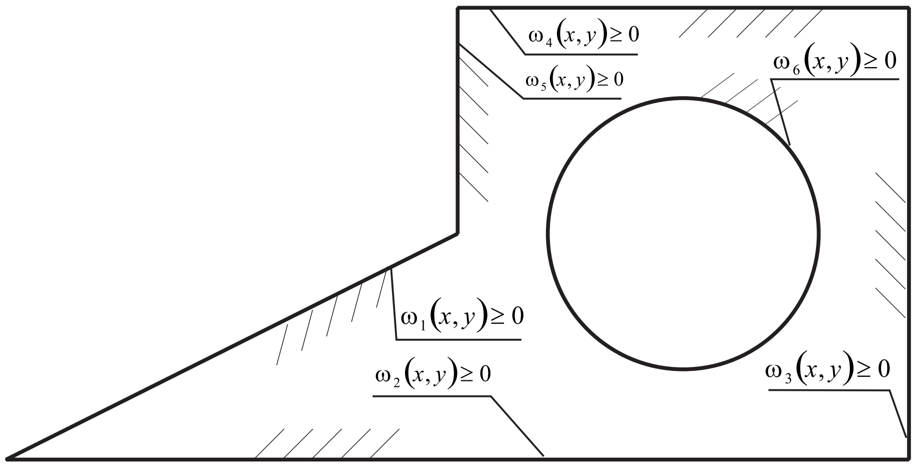
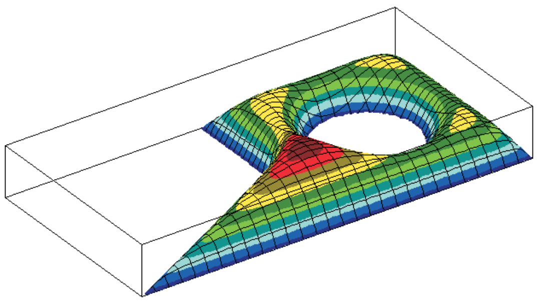


Figure 5: Constructive Solid Geometry represents geometric objects as union and intersection of elementary halfspaces

Rvachev proposed a technique that allows to construct such normalized functions for piecewise smooth boundaries of geometric domains [19, 21, 30]. The constructed functions are differentiable at all interior points of the geometric domain and behave as a distance in the neighborhood of the boundary. The technique proposed by Rvachev is based on the theory of R-functions (see Appendix), which provides the connection between logical and set operations in geometric modeling and analytic constructions. The main result of the theory is that for *every* logical or set-theoretic geometric construction, there is a corresponding real-valued function that implicitly defines the same set of points.


(a)

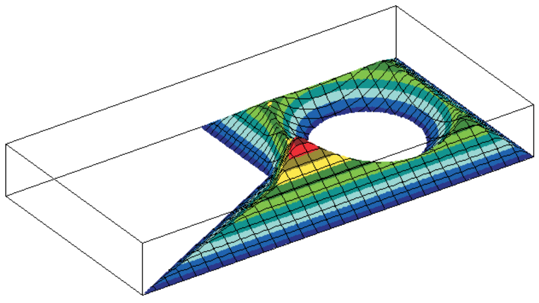

(b)

Figure 6: (a)Approximate distance field constructed from CSG model using R_0-functions; (b)approximate distance field constructed from CSG model using R_p-functions with $p = 16$

Furthermore, the translation from logical and set-theoretic description is a matter of simple syntactic substitution that does not require expensive symbolic computations. For example, the geometric domain in Figure 5 can be defined as a Boolean set combination of six primitives:

$$\Omega = (\omega_1 \cup \omega_5) \cap \omega_2 \cap \omega_3 \cap \omega_4 \cap \omega_6,$$

where the individual primitives ω_1 through ω_6 are defined by the following inequalities:

$$\omega_1 = \frac{0.5x - y}{\sqrt{1.25}} \geq 0; \qquad \omega_2 = y \geq 0; \qquad \omega_3 = -x \geq 0; \qquad \omega_4 = 1 - y \geq 0; \qquad \omega_5 = x - 1 \geq 0;$$

$$\omega_6 = \frac{1}{2}\left((x - 1.5)^2 + (y - 0.5)^2 - 0.25^2\right) \geq 0;$$

This set-theoretic representation can be translated into the function shown in Figure 5 using R-functions:

$$\omega = (\omega_1 \vee_0 \omega_5) \wedge_0 \omega_2 \wedge_0 \omega_3 \wedge_0 \omega_4 \wedge_0 \omega_6, \tag{2}$$

where symbols $\wedge_0$ and $\vee_0$ denote R_0-conjunction and R_0-disjunction (see Appendix and references [21, 30] for more information on R-functions). The function ω is analytic everywhere except the corner points and is normalized on all regular points of the boundary. This example suggests how any Constructive Solid Geometry (CSG) representation of a geometric domain may be translated into the corresponding approximate distance function and Figure 7 diagrams the translation process. Using different systems of R-functions it is possible to control properties of the constructed approximate distance functions. For example, Figure 6(b) presents the approximate distance field which was constructed using R_p system of R-functions with $p = 16$ [30]. This function gives better approximation to the distance function than function shown in Figure 6(a) and is continuously differentiable at the inner points of domain.

Logically, boundary representation of a solid is a union of solid's faces, each face is a subset of some surface bounded by edges, and so on. This logical description can also be directly translated into a function such that it is zero for every point on the boundary and positive elsewhere. Figures 7 and 8 illustrate the construction process of the approximate distance fields from boundary representations: first, the approximate distance functions for trimmed geometrical primitives are constructed [30] (see Figure 8). Then these functions are combined into one function using the appropriate system of R-functions. Analogous arguments hold for general cell complexes and other logical geometric constructions. For additional discussion and examples see reference [30].

2.3 Transfinite Interpolation with Approximate Distance Fields

A typical boundary value problem involves different boundary conditions prescribed on different portions of the boundary. Each such boundary condition induces a separate solution structure, and the solution structure of the boundary value problem must combine all of them together. Formally, given a set of functions prescribed over each portion of the

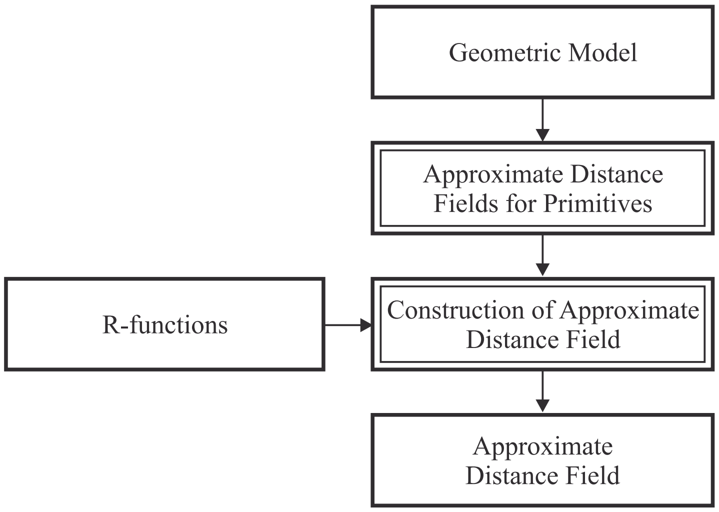

Geometric Model
Approximate Distance Fields for Primitives
R-functions
Construction of Approximate Distance Field
Approximate Distance Field

Figure 7: Construction of the approximate distance fields from geometric models

boundary, we must be able to interpolate them. For the solution process to remain meshfree, the interpolation process should not rely on domain discretization. Since the functions are typically prescribed over continuous boundaries, the interpolation should be performed over an infinite number of points, and therefore such interpolation is usually called "transfinite".

The appendix describes how a classical method for interpolating scattered data by weight functions that are inversely proportional to the distance from the data points can be adopted to perform the transfinite interpolation. The key to the solution of this problem is once again the representation of the boundaries by approximate distance fields, which can be used in place of the exact distances to the individual points. Given a list of pairs $\{\omega_i, f_i\}$, with the approximate distance fields ω_i representing boundary portions where functions f_i are known, the transfinite interpolation of these functions can be performed by the following recursive algorithm:

```
Interpolate( list_of_pairs L );
{
    $f_1$ = L.GetFunction(1);
    $\omega_1$ = L.GetDistance(1);
    For i=2 to L.GetSize()
    {
        $f_2$ = L.GetFunction(i);
        $\omega_2$ = L.GetDistance(i);
        $f_1 = (f_1\omega_2 + f_2\omega_1)/(\omega_1 + \omega_2)$;
        $\omega_1 = \omega_1 \wedge_0 \omega_2$;
    }
    return $f_1$;
}
```

The list of pairs `L` is implemented as a class which contains functions `GetDistance(i)`, `GetFunction(i)` and `GetSize()`. Function `GetDistance(i)` gives the approximate distance field from i-th geometric object. Function `GetFunction(i)` returns the function defined over i-th geometrical object. Function `GetSize()` gives the number of elements in the list. Every time the algorithm advances along the list, it recursively constructs the interpolating function and the approximate distance function to all previously visited geometrical objects. Figure 9(b) shows a function interpolating the values prescribed at the boundaries of geometric domain as shown in Figure 9(a).

The same method of transfinite interpolation may be employed for interpolation of values, derivatives, or complete solution structures [24]. Thus, we can interpolate individual boundary conditions to construct the interpolating part

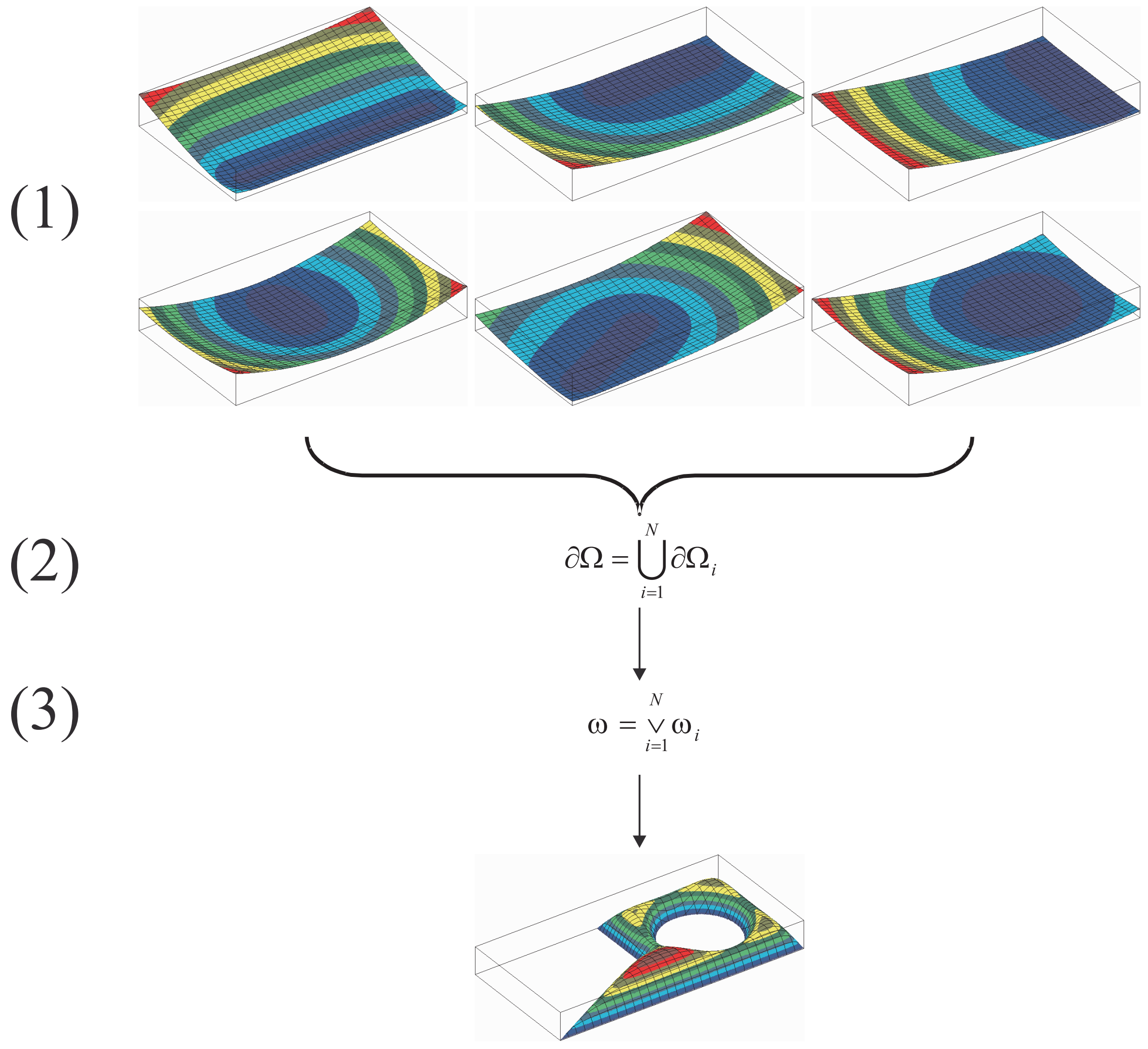
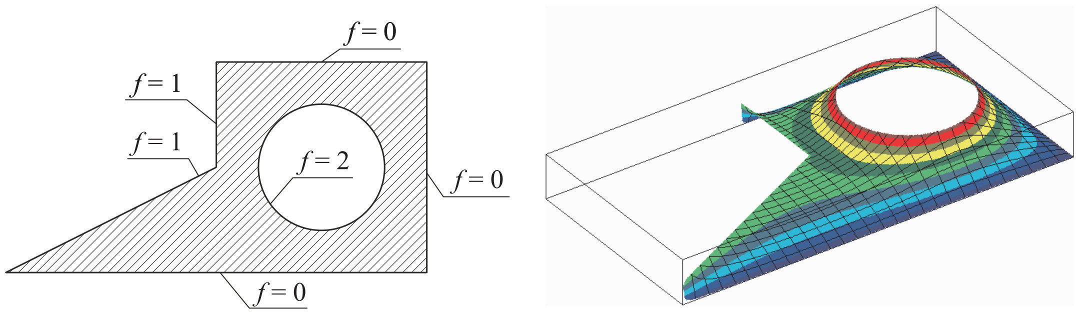

(1)
(2)
$\partial\Omega = \bigcup_{i=1}^{N} \partial\Omega_i$
(3)
$\omega = \bigvee_{i=1}^{N} \omega_i$

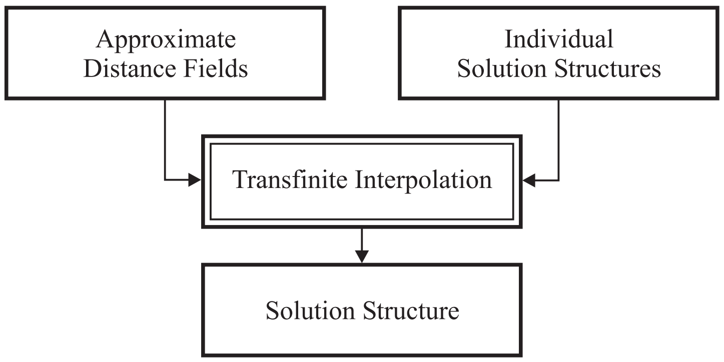
Figure 8: Construction of the approximate distance fields from geometric models presented as Boundary Representation

of the solution structure and then add the approximating terms as appropriate; alternatively, individual boundary conditions may be grouped according to their type, then individual solution structures are constructed for each type of the boundary conditions, and these solution structures are transfinitely interpolated into a solution structure which satisfies all given boundary conditions (see Figure 10).



(a) (b)

Figure 9: (a) Geometric domain with prescribed boundary conditions; (b) function interpolating the given boundary conditions



Approximate Distance Fields
Individual Solution Structures
Transfinite Interpolation
Solution Structure

Figure 10: RFM solution structure can be constructed by transfinite interpolation of solution structures constructed for individual boundary conditions

3 Computational Tools in SAGE

As Figures 2 and 3 show, approximating the solution to a boundary value problem requires differentiating the solution structure, integrating over the geometric domain, solving a system of algebraic equations, and visualizing of the results. Except for the solution of the algebraic equations, the required computational procedures are substantially different from those employed in mesh-based methods. Below we describe each of these modules as they have been implemented in SAGE.

3.1 Automatic Differentiation at Run Time

A typical meshfree solution procedure requires computing partial derivatives of the solution structure, for example with respect to unknown coefficients and/or other parameters. The difficulty is that the solution structure depends on the domain-dependent distance fields that are not known a priori. This means that differentiation must take place at run time – with a precision and speed that would support typical engineering analysis. Many differentiation methods have been developed, but all of them can be divided into three categories: numerical, symbolic, and automatic differentiation methods. Numerical differentiation techniques are not sufficiently accurate for most applications especially when high order derivatives have to be computed. Symbolic differentiation methods aim at derivation of symbolic expressions for the derivatives. For example a symbolic differentiation in Mathematica [36] of the function

$$f(x,y) = e^{-(x^2+y^2)}\cos(x)\cos(y) \tag{3}$$

(its plot is shown in Figure 11(a)) results in the following expression for the partial derivative $\frac{\partial^2 f(x,y)}{\partial x \partial y}$: $\frac{\partial^2 f(x,y)}{\partial x \partial y} = e^{-(x^2+y^2)}(4xy\cos(x)\cos(y) + 2y\sin(x)\cos(y) + 2x\cos(x)\sin(y) + \sin(x)\sin(y))$. Numerical values of a derivative are obtained via evaluation of the symbolic expression at the given point:

$$\begin{aligned}\frac{\partial^2 f(0.2,0.3)}{\partial x \partial y} =& e^{-(0.2^2+0.3^2)}(4\cdot 0.2\cdot 0.3\cos(0.2)\cos(0.3) + 2\cdot 0.3\sin(0.2)\cos(0.3)+ \\ & 2\cdot 0.2\cos(0.2)\sin(0.3) + \sin(0.2)\sin(0.3)) \approx 0.450595.\end{aligned} \tag{4}$$

Symbolic differentiation methods can compute the numerical values of the derivatives with high accuracy, but they face insurmountable computational difficulties when functions to be differentiated grow in size or are constructed by algorithms.

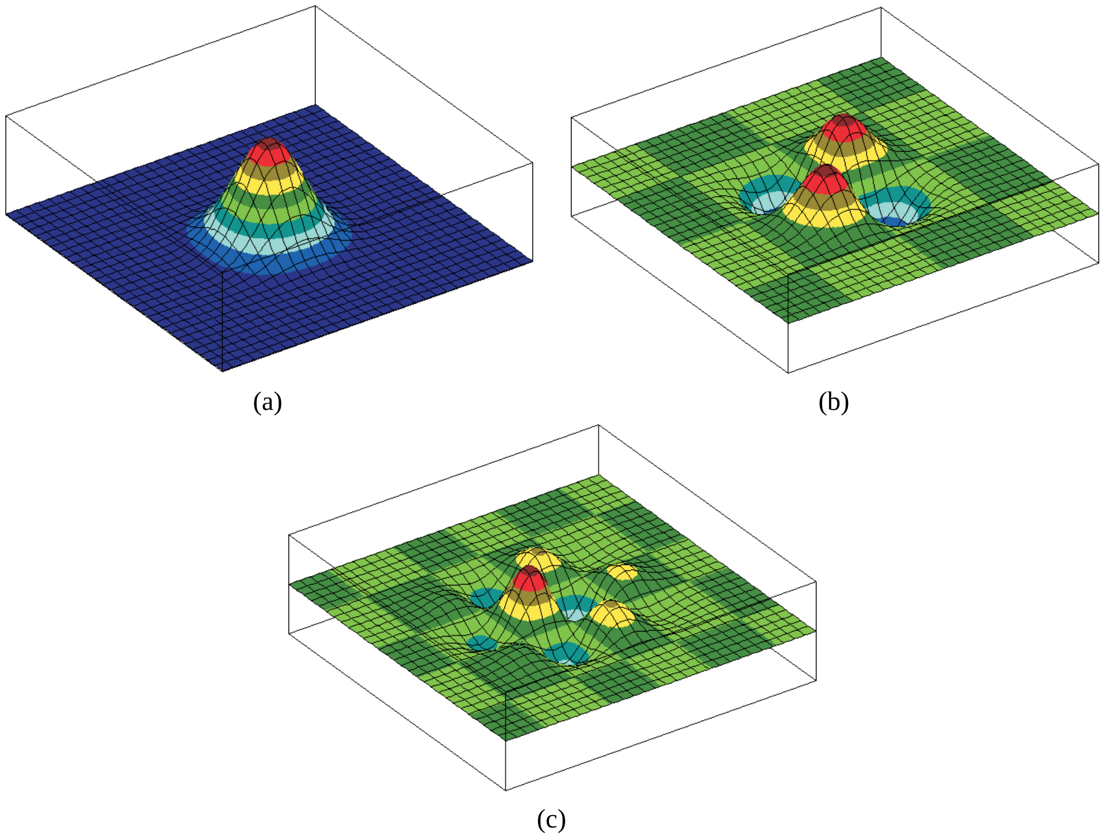

(a) (b) (c)

Figure 11: (a)Plot of the function $f(x,y) = e^{-(x^2+y^2)}\cos(x)\cos(y)$ for $x \in [-\pi,\pi]$ and $y \in [-\pi,\pi]$; (b) plot of $\frac{\partial^2 f(x,y)}{\partial x \partial y}$ for $x \in [-\pi,\pi]$ and $y \in [-\pi,\pi]$; (c) plot of $\frac{\partial^5 f(x,y)}{\partial x^3 \partial y^2}$ for $x \in [-\pi,\pi]$ and $y \in [-\pi,\pi]$

Automatic differentiation methods have been developed to address these computational difficulties. Automatic differentiation employs the same exact differentiation rules, but propagates the numerical values of the derivatives rather than their symbolic expressions. This allows computing derivatives of any order with high accuracy up to round-off error, as well as differentiation of functions constructed by algorithms or represented by a computer program [2, 6, 22, 33, 34].

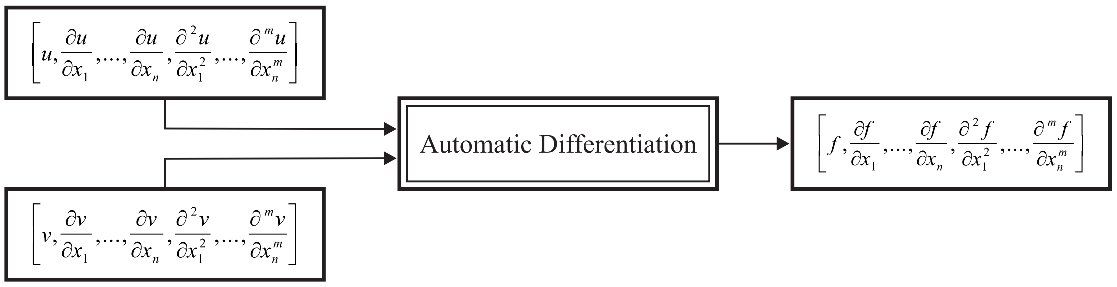

$\left[u, \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial x_1^2}, \ldots, \frac{\partial^m u}{\partial x_n^m}\right]$
$\left[v, \frac{\partial v}{\partial x_1}, \ldots, \frac{\partial v}{\partial x_n}, \frac{\partial^2 v}{\partial x_1^2}, \ldots, \frac{\partial^m v}{\partial x_n^m}\right]$
Automatic Differentiation
$\left[f, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}, \frac{\partial^2 f}{\partial x_1^2}, \ldots, \frac{\partial^m f}{\partial x_n^m}\right]$

Figure 12: Parameters and the result of any Automatic Differentiation procedure are tuples of derivatives

The procedure is illustrated in Figure 12. Given a point in space, automatic differentiation computes all partial derivatives up to the specified order at that point. Derivatives are computed in forward mode — from independent to dependent variables. All derivatives of independent variables are zero except the first partial derivative with respect to that independent variable, which has value of unity. This procedure is applied recursively to any graph representation of a function, such as those constructed by most compilers. At each node of the graph, derivatives are transformed according to the corresponding differentiation rules that are known for all elementary functions and arithmetic operations [22, 33, 16]. For example, Figure 13 illustrates the automatic differentiation of function (3) at the point (0.2,0.3). Figures 11(b) and(c) give plots of the partial derivatives $\frac{\partial^2 f(x,y)}{\partial x \partial y}$ and $\frac{\partial^5 f(x,y)}{\partial x^3 \partial y^2}$ of function (3) for $x \in [-\pi, \pi]$ and $y \in [-\pi, \pi]$.

The ordered sequences of functions and their partial derivatives shown in Figure 12 and 13 are called *tuples*. In SAGE, automatic differentiation is implemented using *tuple* class which overloads the usual type of function's value. Overloaded arithmetic operators and elementary functions provide a friendly interface for users of the class. Consequently, in SAGE, tuple-value functions processed by automatic differentiation appear as the usual C++ functions, but they also handle the derivative data – simultaneously and transparently. For example, the following C++ code performs automatic differentiation of function (3):

```
tuple f(double x, double y)
{
    tuple X, Y, F;
    Argument(X, 1, x);
    Argument(Y, 2, y);
    F = exp(-(square(X)+square(Y)))*cos(X)*cos(Y);
    return F;
}
```

Function `Argument` generates derivatives for independent variables, and the numbers "1" and "2" correspond to x and y independent variables respectively. Following the call of function `Argument` variables `X` and `Y` hold the values and derivatives of the corresponding independent variables. Then these variables are passed into expression `F=exp(-(square(X)+square(Y)))*cos(X)*cos(Y)` that comprises overloaded functions. At the end of the computation, variable `F` contains values of all partial derivatives of the function up to the specified order.

3.2 Adaptive Integration Over the Non-meshed Domain

All numerical integration methods represent the integrals as weighted sum of the function's values computed at selected "integration points". In the context of boundary value problems, integration of functions and/or their derivatives must take place over the geometric domain represented in a geometric modeling system. For reasonably complex domains, the integration process is carried out as a collection of local integrations over simple subsets of the domain. The most common decomposition is a uniform orthogonal non-conforming mesh of space into rectangular cells. Integrat-

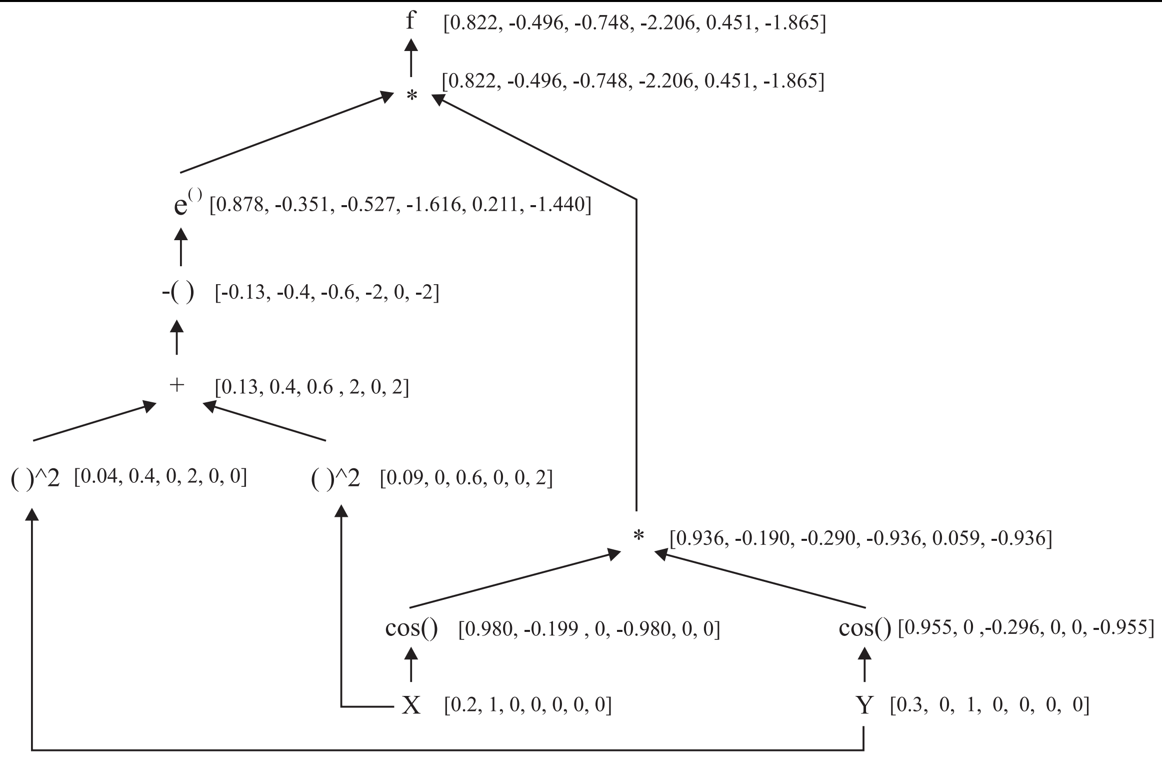

f [0.822, -0.496, -0.748, -2.206, 0.451, -1.865]
* [0.822, -0.496, -0.748, -2.206, 0.451, -1.865]
e^( ) [0.878, -0.351, -0.527, -1.616, 0.211, -1.440]
-( ) [-0.13, -0.4, -0.6, -2, 0, -2]
+ [0.13, 0.4, 0.6 , 2, 0, 2]
( )^2 [0.04, 0.4, 0, 2, 0, 0]
( )^2 [0.09, 0, 0.6, 0, 0, 2]
* [0.936, -0.190, -0.290, -0.936, 0.059, -0.936]
cos() [0.980, -0.199 , 0, -0.980, 0, 0]
cos() [0.955, 0 ,-0.296, 0, 0, -0.955]
X [0.2, 1, 0, 0, 0, 0, 0]
Y [0.3, 0, 1, 0, 0, 0, 0]

Figure 13: Automatic differentiation of the function $f(x, y) = e^{-(x^2+y^2)} \cos(x) \cos(y)$is performed in forward mode — from independent to dependent variables

ing cell-by-cell implements the divide-and-conquer paradigm for integration of global functions such as polynomials, and facilitates efficient integration of functions with local support, such as B-splines. As shown in Figure 14, some cells intersect the boundary of the domain and some do not. A hierarchical decomposition of space such as shown in Figure 15 for the domain in Figure 5 accommodates variable size cells: larger in the interior of the domain and smaller near the boundary. This distribution assures that the portion of the geometric domain enclosed in each integration cell is simply connected. Similar decomposition is often called a quadtree in 2D and octree in 3D, and they are easily computed by most geometric modelers.

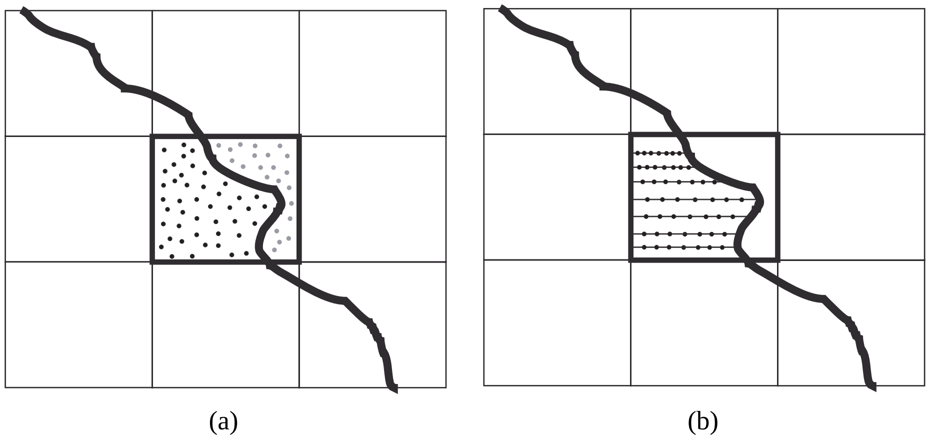


Figure 14: (a)Monte Carlo integration methods use randomly distributed integration points; (b)Gaussian integration methods allocate integration points inside domain

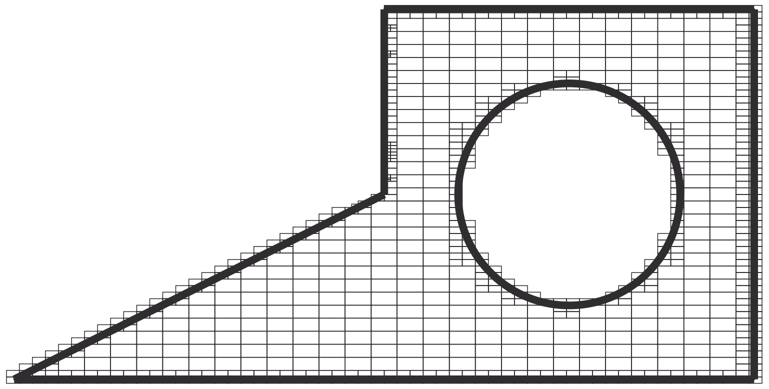


Figure 15: Hierarchical space decomposition of the domain shown in Figure 5

The remaining non-trivial task is the integration of an a priori unknown function over the cells of the non-conforming mesh. Various numerical integration methods have been developed for integration of functions of one independent variable. But the integration of functions of two and more independent variables remains a challenging problem for all meshfree methods, because the integration domain (the cell) can have an arbitrary shape.

Numerical integration methods differ from each other by values of weights and positions of the integration points [17]. The simplest integration method is the Monte Carlo technique that is rooted in the probability theory. The function is sampled at randomly distributed integration points as it is shown in Figure 14(a). The value of the integral is determined as the ratio of sum of the function's values at the interior points to the total number of sampled points. The advantages of the Monte Carlo integration method is that it does not need advanced geometric computations beyond the standard test to determine if the integration point is inside or outside the integration domain. It is also easily adapted for integration over domains with arbitrary shape and topology in n-dimensional space [17]. Although several adaptive Monte Carlo integration methods are known [17], all of them exhibit slow convergence and require relatively large numbers of the integration points in comparison with other integration methods.

The most accurate among the known integration methods are methods based on Gaussian quadrature rules [17]; for example, in the case of one independent variable, polynomials of degree up to $2n - 1$ can be integrated exactly by sampling the function at n integration points. Integration in two and higher dimensions invokes successive application of one dimensional integration rules. This is straightforward for rectangular cells that are contained completely inside the domain, but the choice of good integration points is not clear inside those cells that intersect the domain's boundary. For such cells, numerical values of the weights and the positions of the integration points also depend on the shape of the integration domain.

Figure 14(b) illustrates one possible application of Gaussian rule that considers the boundary cell to be a horizontal deformation of the rectangular cell. This idea, originally proposed in [27], led us to develop a general approach to integration that combines the Gauss quadrature with the well known technique of "marching cubes" in computer graphics [37]. Every cell intersecting the boundary of the domain falls into one of the four types shown in Figure 16. In each case, the cell is considered a deformation of a rectangular cell parameterized by rays that are aligned with coordinate axes. The geometric modeler computes the intersection of rays with the boundary of the domain, and the integration points are allocated along the rays as shown in Figure 16. A similar technique works for integrating over three-dimensional solid domains, with a somewhat larger number of special cases.

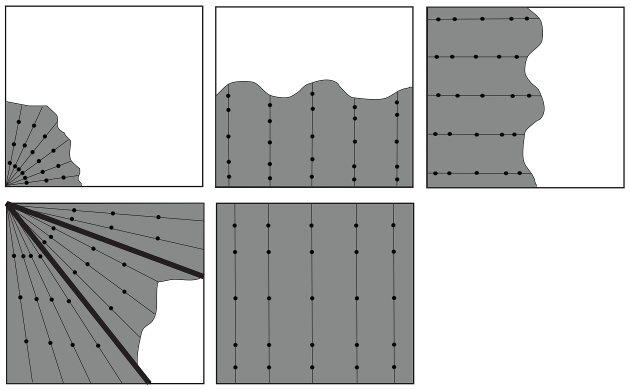


Figure 16: Allocation of the integration points depends on the type of intersection boundary with a cell

Integration errors can be estimated using using a posteriori analysis with Gauss-Kronrod quadratures [11]. If the integration accuracy is unsatisfactory the integration algorithm performs adaptive refinement – either to the geometry of the domain by further subdivision of the integration cells, or to the integrand by increasing the number of integration points. Figure 17 shows the locations of all integration points inside the geometric domain of Figure 5.

3.3 Field Visualization

The final step of the solution procedure is the visualization of the computed results in the form of a field $f(x, y, z, \ldots)$ (scalar or vector) over the geometric domain. Visualization of fields over meshed domains is a common task that usually relies on color-coded isolines and/or isosurfaces or graphs in the specified sections, but difficulty arises once again because the original geometric domain has not been meshed. In this sense the visualization problem appears to be similar to the numerical integration problem, and indeed a background mesh of space similar to the integration mesh can be used to sample and visualize the field either in 2D or 3D.

Visualization of fields over three-dimensional domains can also be reduced to a two-dimensional visualization problem over the domain's boundaries. Modern solid modeling systems routinely generate triangulations of boundary representations that are well suited for visualization purposes. These triangulations can be recursively and adaptively refined to capture features of interest in the geometric domain and/or field properties. The field is then sampled on the vertices of the triangulation and interpolated inside each triangle. Figure 18(b) shows an example of a temperature

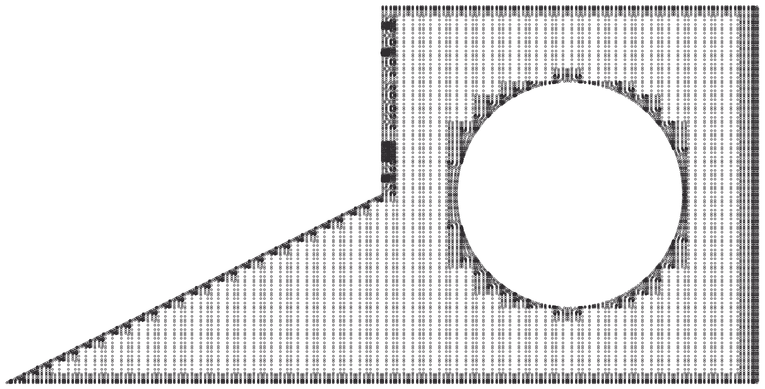
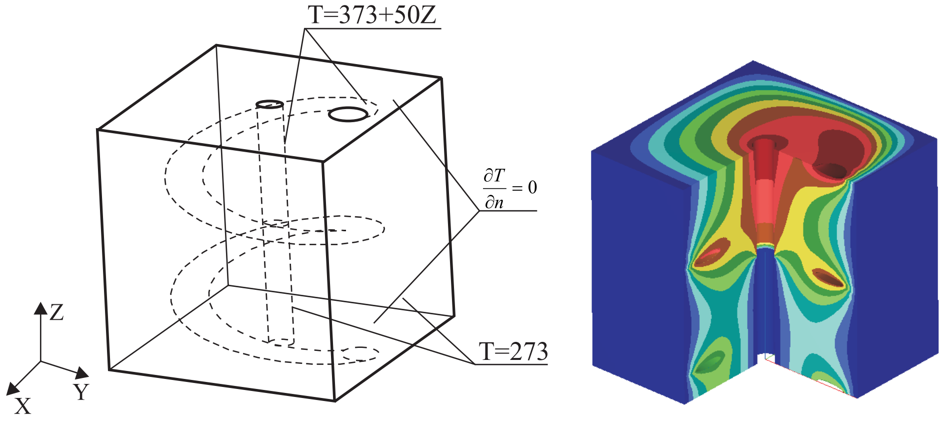


Figure 17: Locations of the integration points inside the domain shown in Figure 5

field visualized over the solid shown in Figure 18(a) using this technique.



(a) (b)

Figure 18: (a) A 3D geometric model with assigned boundary conditions and (b) distribution of the temperature field inside the geometric object

4 Conclusions

4.1 Towards Integration of Geometric and Field Modeling

We described the architecture of a particular meshfree system, SAGE, intended specifically for computing solutions to boundary value problems using RFM. However, the described methods, techniques, and algorithms are applicable to all mesh-based and meshfree methods and have broad applicability to field problems beyond boundary value problems. Examples of such applications include:

- Approximate distance fields have many applications in computer graphics, geometric modeling, and engineering applications. For instance, the automatically constructed fields may be used to construct potential functions in robot motion planning problems, material distributions for parts with functionally graded materials, enrichment functions for a priori known solutions, or level set surfaces.

- Transfinite interpolation with inverse distance weighting is extremely useful in computer-aided geometric design where the interpolated function represents the geometry of the surface, in material modeling where distinct material features must be interpolated in a smooth manner, and in other applications where exact satisfaction of boundary conditions is essential.
- The automatic differentiation library puts no restrictions on number of independent variables or the order of the derivatives. It can be used to differentiate programs, known functions, or functions constructed at run time – efficiently and accurately. Besides meshfree analysis the automatic differentiation has numerous applications in sensitivity analysis, solving ordinary differential equations, control and other engineering problems. The Fast Forward Automatic Differentiation Library (FFADLib) can be downloaded from http://sal-cnc.me.wisc.edu.
- Adaptive integration and visualization modules can be used for various applications involving fields over geometric domains represented by solid models.

The modular design of SAGE supports straightforward integration with other systems. SAGE relies only on generic geometric computations that can be performed by virtually any geometric modeling kernel or library. The implemented modules can also be combined with most classical solution methods, including finite elements and finite differences.

As a general purpose system for meshfree modeling and analysis, SAGE can be used to tackle almost any boundary value problem. The approach is firmly rooted in sound theory guaranteing completeness in the classical sense [23], and experimental evidence suggests a faster convergence to the solution when the boundary conditions are satisfied exactly. Our description of the method and the algorithms in this paper is necessarily incomplete and was designed for maximum conceptual clarity; many non-trivial improvements are known and have been implemented by the authors and others. For example, recently Hollig and his coworkers have optimized RFM with B-splines as basis functions into the method of WEB-splines that relies on local computations, multi-resolution, and leads to better conditioned algebraic systems [9, 7]. They also provide estimates on the rate of convergence for this particular version of the method.

For most engineering applications, SAGE or another meshfree computational environment with a similar architecture, offers numerous advantages in comparison to the more traditional engineering analysis systems. Independence from a mesh eliminates the need for conformal meshing and supports complete automation, effortless changes, motions and deformations, parametric studies, shape optimization, multi-physics, and time-varying boundary conditions. A fully functional two-dimensional prototype of SAGE that solves heat transfer and plate vibration problems on multiply-connected polygonal domains may be downloaded freely from http://sal-cnc.me.wisc.edu. We conclude with additional illustrative examples of engineering applications successfully solved with SAGE.

4.2 Example Applications

Heat transfer Figure 18(b) shows the computed temperature field inside the geometric domain with boundary conditions prescribed as in Figure 18(a). Homogeneous Neumann boundary conditions are indicated on the top and bottom faces of the cube, and non-homogeneous Dirichlet boundary conditions are specified on other portions of the boundary of the domain (see Figure 18(a)). Accordingly, the solution structure for mixed boundary conditions (shown in Table 1) is selected. The approximate three-dimensional distance fields for portions of the boundaries were constructed directly from a CSG representation of the geometric domain. The basis functions $\{\chi_i\}_{i=1}^N$ in the solution structure have been chosen to be tensor products of one dimensional cubic B-splines defined over the uniform orthogonal grid $31 \times 31 \times 31$. Numerical values of the unknown coefficients $\{C_i\}_{i=1}^N$ in the solution structures were computed using the Ritz method, with integration performed as described in Section 3.2. For visualization purposes, the computed field was sampled on a $90 \times 90 \times 90$ uniform orthogonal grid and then tri-linearly interpolated within each cell. All geometric computations were performed by the Parasolid solid modeling kernel.

Plate vibration Figure 20 shows the first four natural vibration modes of the thin plate shown in Figure 19(a). The plate is fixed at its boundary points, preventing both deflection and rotation. An appropriate solution structure for this boundary value problem is derived in [26] as

$$u = \sum_{i=1}^{N} C_i \omega^2 \chi_i$$

The approximate distance function ω, whose plot is shown in Figure 19(b), was constructed automatically from a CSG representation of the geometric domain as described in Section 2.2. In this case, the basis functions $\{\chi_i\}_{i=1}^N$ in the solution structure were selected from polynomials of degree ≤ 9. The solution method described in [26] reduces solution of the biharmonic equation to the generalized algebraic eigenvalue problem. The integration was performed using the six point Gauss-Legendre quadrature on a hierarchical non-conforming decomposition of space as described in Section 3.2. The initial integration grid was 11×11 with one more level of subdivision applied to each cell intersecting the boundary of the domain. A uniform Cartesian 100×100 grid and bilinear interpolation were used to produce the pictures in Figures 19 and 20.

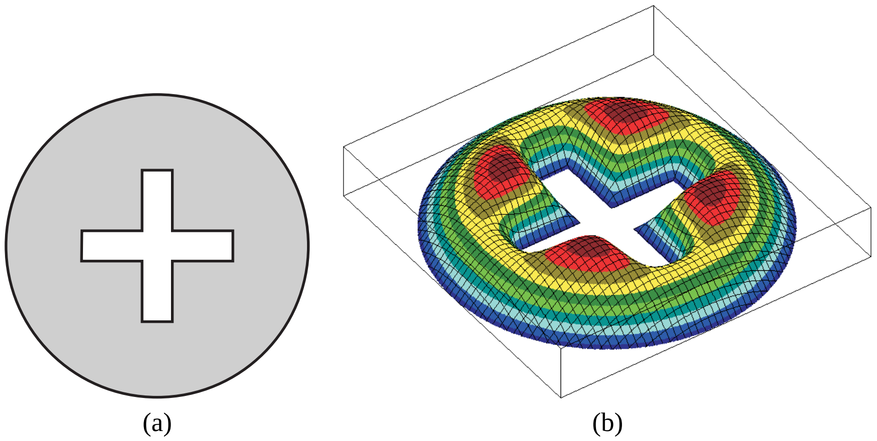


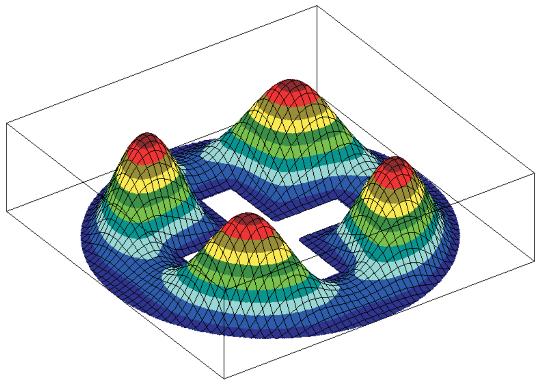
Figure 19: (a) A thin plate with fixed boundary; (b) approximate distance field

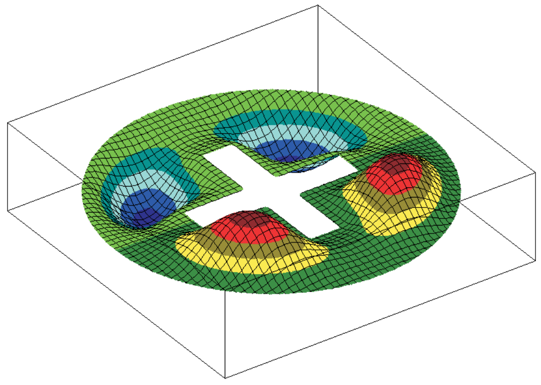
Cantilever structural analysis Application of SAGE to solve vector-valued field problems is a straightforward extension employing a vector of solution structures, with each scalar component represented by its own scalar solution structure. Figure 21(a) shows a loaded cantilever structure. The computed distribution of the principal stress σ_1 is shown in Figure 21(b). Each component of the displacement vector is represented by its own solution structure satisfying the kinematic boundary conditions [21]:

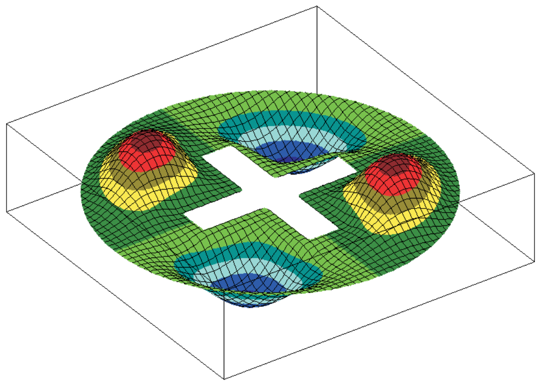
$$u = \sum_{i=1}^{N} C_i^u \omega \chi_i; \qquad v = \sum_{i=1}^{N} C_i^v \omega \chi_i,$$

where $\omega = \omega(x, y)$ measures approximate distance from the circular holes where $u = 0$ and $v = 0$ (Figure 21(a)). This vector solution structure utilized bicubic B-splines defined over 70×70 uniform Cartesian grid. The solution was obtained by minimizing the potential energy of the cantilever structure. Numerical integration was performed over the non-conforming grid of B-splines, with 4×4 integration points in each cell, and each boundary cell was subdivided into four subcells. The computed field in plotted using bi-linear interpolation of the data sampled on a 200×200 grid.

Incompressible fluid flow SAGE has also been applied successfully to solve non-linear vector problems, in particular incompressible fluid dynamics problems. Figure 23 shows distributions of the horizontal component of the velocity vector and the pressure field around a car inside a wind tunnel (Figure 22). This solution was obtained using artificial compressibility approach, approximating the velocity components by solution structures that satisfy the prescribed velocity profile (parabolic for u component, and zero for v component of the velocity vector) at the inlet, non-slip boundary conditions at the walls of the tunnel and at the car body, and homogeneous Neumann boundary conditions at the outlet [35]. The solution structure for each component of the velocity vector and pressure utilized bi-cubic B-splines over 120×60 uniform rectangular grid. The approximate distance fields were constructed from the boundary representation following the method described in Section 2.2. Numerical integration was performed over the grid of B-splines, with extra subdivision applied to each boundary cell, and 4 integration points in each coordinate direction.


(a)


(b)


(c)

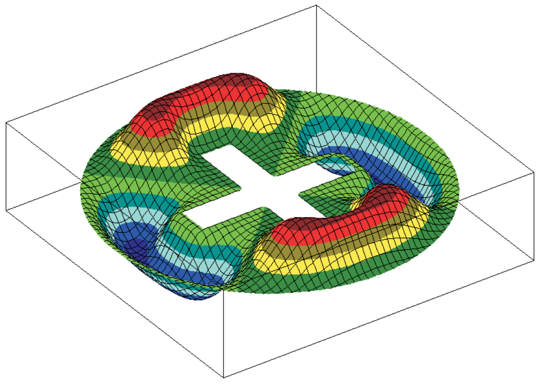
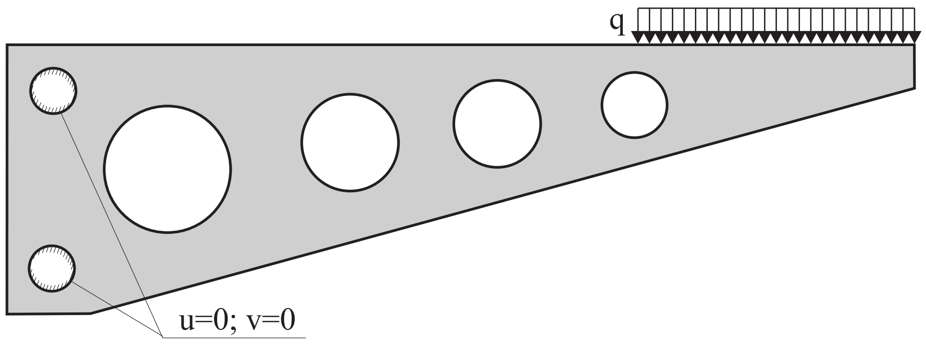
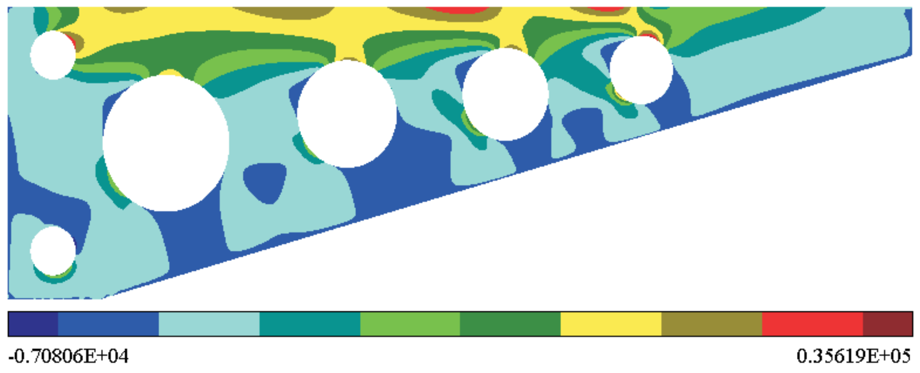

(d)

Figure 20: Natural vibration modes of the plate shown in Figure 19



(a)



(b)

Figure 21: (a) Loaded cantilever beam and (b) distribution of the principal stress σ_1

Results were plotted on the 200×200 visualization grid with bilinear interpolation. Extensive numerical experiments, as well as comparisons with experimental data and numerical results obtained by other methods can be found in [35].

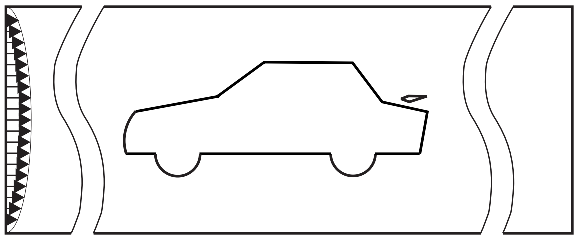


Figure 22: Car in a wind tunnel

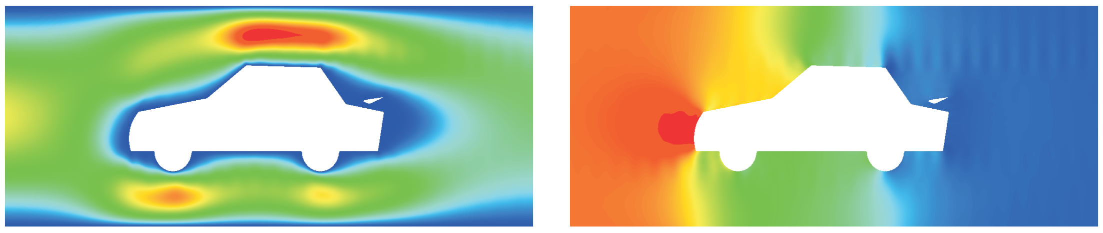


Figure 23: Distribution of (a) the horizontal component of the velocity vector and (b) pressure around the car shown in Figure 22

Acknowledgments

This research is supported in part by the National Science Foundation grants DMI-9900171, DMI-0115133, and CCR-0112758, General Motors Corporation, and the NATO Linkage grant PST.CLG.976192.

References

[1] S.N. Atluri and T. Zhu. A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics. *Computational Mechanics*, 22(2):117–127, 1998.

[2] Martin Berz. The DA precompiler DAFOR. Tech. Report, Lawrence Berkeley National Laboratory, Berkeley, Calif., 1990.

[3] J. Bloomenthal. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, 1997.

[4] J.-S. Chen, C. Pan, C.M.O.L. Roque, and H.-P. Wang. A lagrangian reproducing kernel particle method for metal forming analysis. *Computational Mechanics*, 22:289–307, 1998.

[5] C. A. Duarte and J. T. Oden. An h-p adaptive methods using clouds. *Computer Methods in Applied Mechanics and Engineering*, 139:237–262, 1996.

[6] A. Griewank, D. Juedes, H. Mitev, J. Utke, O. Vogel, and A. Walther. ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++. *ACM TOMS*, 22(2)(June):131–167, 1996. Algor. 755.

[7] K. Hollig. *Finite Element Methods with B-Splines*. SIAM, 2002. In press.

[8] J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. A K Peters, 1993.

[9] Hollig K., Reif U., and Wipper J. Weighted extended b-spline approximation of dirichlet problems. *SIAM Journal on Numerical Analysis*, 39(2):442–462, 2001.

[10] L. V. Kantorovich and V. I. Krylov. *Approximate Methods of Higher Analysis*. Interscience Publishers, 1958.

[11] A. S. Kronrod. *Nodes and Weights of Quadrature Formulas*. Cosultants Bureau, 1965.

[12] W. K. Liu, S. Jun, S. Li, J. Adee, and T. Belytschko. Reproducing kernel particle methods for structural mechanics. *Int. J. Numer. Methods Engrg.*, 38:1655–1679, 1995.

[13] L. Lucy. A numerical approach to testing the fission hypothesis. *Astron. J.*, 82:1013–1024, 1977.

[14] J. M. Melenk and I. Babuska. The partition of unity finite element method: Basic theory and applications. *Computer Methods in Applied Mechanics and Engineering*, 139:289–314, 1996.

[15] B. Nayroles, G. Touzot, and P. Villon. Generalizing the finite element method: diffuse approximation and diffuse elements. *Computational Mechanics*, 10:307–318, 1992.

[16] Richard D. Neidinger. An Efficient Method for the Numerical Evaluation of Partial Derivatives of Arbitrary Order. *ACM Transactions on Mathematical Software*, 18(2):159–173, 1992.

[17] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

[18] P. W. Randles and L. D. Libersky. Smoothed particle hydrodynamics: Some recent improvements and applications. *Computer Methods in Applied Mechanics and Engineering*, 139:375–408, 1996.

[19] V. L. Rvachev. Analytical description of some geometric objects. *Dokl AS USSR*, 153(4):765–768, 1963.

[20] V. L. Rvachev. *Geometric Applications of Logic Algebra*. Naukova Dumka, 1967. In Russian.

[21] V. L. Rvachev. *Theory of R-functions and Some Applications*. Naukova Dumka, 1982. In Russian.

[22] V. L. Rvachev, G. P. Manko, and V. V. Fedko. To the problem of software engineering of the computer differentiation of superpositions of the functions. *Dokl AS UkrSSR*, (1):72–74, 1981. In Russian.

[23] V. L. Rvachev, T. I. Sheiko, V. Shapiro, and I. Tsukanov. On Completeness of RFM Solution Structures. *Computational Mechanics*, 22(1), 2000.

[24] V. L. Rvachev, T. I. Sheiko, V. Shapiro, and I. Tsukanov. Transfinite interpolation over implicitly defined sets. *Computer Aided Geometric Design*, 18(4):195–220, 2001.

[25] V. L. Rvachev and A. N. Shevchenko. *Problem-oriented languages and systems for engineering computations*. Tekhnika, Kiev, 1988. In Russian.

[26] V.L. Rvachev and L.V. Kurpa. *R-functions in problems of the theory of plates*. Nauk. dumka, Kiev, 1987. In Russian.

[27] V.L. Rvachev, A.N. Shevchenko, and V.V. Veretel'nik. Numerical integration software for projection and projection-grid methods. *Cybernetics and Systems Analysis*, 30(1):154–158, 1994.

[28] V. Shapiro. Well-formed set representations of solids. *International Journal on Computational Geometry and Applications*, 9(2):125–150, 1999.

[29] V. Shapiro. *Handbook of Computer Aided Geometric Design*, chapter Solid Modeling. Elsevier Science, 2002.

[30] V. Shapiro and I. Tsukanov. Implicit functions with guaranteed differential properties. In *Fifth ACM Symposium on Solid Modeling and Applications*, Ann Arbor, MI, 1999.

[31] V. Shapiro and I. Tsukanov. Meshfree simulation of deforming domains. *Computer-Aided Design*, 31(7):459–471, 1999.

[32] D. Shepard. A two-dimensional interpolation function for irregularly spaced data. In *Proceedings 23rd ACM National Conference*, pages 517–524, 1968.

[33] A. N. Shevchenko. DIFOR and its application for automation of programming of boundary value problems. Tech. report 32, Institute for Problems in Machinery of Ukrainian Academy of Sciences, Kharkov, Ukraine, 1977.

[34] I. Tsukanov and M. Hall. Fast Forward Automatic Differentiation Library (FFADLib): A User Manual. Tech. report 2000-4, Spatial Automation Laboratory, http://sal-cnc.me.wisc.edu, 2000.

[35] I. Tsukanov, V. Shapiro, and S. Zhang. A meshfree method for incompressible fluid dynamics problems. Tech. report 2002-1, Spatial Automation Laboratory, http://sal-cnc.me.wisc.edu, 2001.

[36] S. Wolfram. *The Mathematica. Fourth edition.* Wolfram Media, 1999.

[37] B. Wyvill, C. McPheeters, and B. Wyvill. Data structure for *soft* objects. *The Visual Computer*, 2(4):227–234, August 1986.

Appendix

The Theory of R-functions

An R-function is a real-valued function whose sign is completely determined by the signs of its arguments. For example, the function xyz can be negative only when the number of its negative arguments is odd. A similar property is possessed by functions $x+y+\sqrt{xy+x^2+y^2}$ and $xy+z+|z-yx|$, and so on. Such functions 'encode' Boolean logic functions and are called ***R*-functions**. Every Boolean function is a companion to infinitely many R-functions, which form a *branch* of the set of R-functions. For example, it is well known that $\min(x_1, x_2)$ is an R-function whose companion Boolean function is logical "and" ($\wedge$), and $\max(x_1, x_2)$ is an R-function whose companion Boolean function is logical "or" ($\vee$). But the same branches of R-functions contain many other functions, e.g.

$$\begin{aligned} x_1 \wedge_\alpha x_2 &\equiv \tfrac{1}{1+\alpha}\left(x_1+x_2-\sqrt{x_1^2+x_2^2-2\alpha x_1 x_2}\right); \\ x_1 \vee_\alpha x_2 &\equiv \tfrac{1}{1+\alpha}\left(x_1+x_2+\sqrt{x_1^2+x_2^2-2\alpha x_1 x_2}\right), \end{aligned} \tag{5}$$

where $\alpha(x_1, x_2)$ is an arbitrary function such that $-1 < \alpha(x_1, x_2) \leq 1$. The precise value of α may or may not matter, and often it can be set to a constant. For example, setting $\alpha = 1$ yields the functions $\min$ and $\max$ respectively, but setting $\alpha = 0$ results in much nicer functions $\vee_0$ and $\wedge_0$ that are *analytic* everywhere except when $x_1 = x_2 = 0$. Many other systems of R-functions are studied in [21]. The choice of an appropriate system of R-functions is dictated by many considerations, including simplicity, continuity, differential properties, and computational convenience.

Just as Boolean functions, R-functions are closed under composition. Using R-functions, any object defined by a predicate on "primitive" geometric regions (e.g. regions defined by a system of inequalities) can now also be represented by a *single* function ω. The latter can be evaluated, differentiated, and possesses many other useful properties.

Transfinite Interpolation

The transfinite interpolation employs the inverse distance weighting technique which was used by Shepard [32] for interpolation of meteorological and geographical/geological data. He interpolated the data defined over discrete set of points. Rvachev generalized this method for interpolation of the functions given over the geometrical objects whose boundary is described by equation $\omega_i(\mathbf{x}) = 0$ [20]. In his original implementation Shepard used Euclidean distances to the points. However, all properties of the inverse distance weighting technique are preserved if it employs the approximate distance functions $\omega_i = \omega_i(\mathbf{x})$ [24]. According to Rvachev the interpolating function appears as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x}) W_i(\mathbf{x}), \tag{6}$$

where each weight function W_i is inversely proportional to the distance from the boundary of i-th geometrical object where the function $f_i(\mathbf{x})$ is prescribed. The weight functions W_i, $(i = 1, ..., n)$ are constructed by normalizing each inverse distance:

$$W_i\left(\mathbf{x}\right) = \frac{\omega_i^{-\mu_i}\left(\mathbf{x}\right)}{\sum\limits_{j=1}^{n} \omega_j^{-\mu_j}\left(\mathbf{x}\right)}. \tag{7}$$

The exponents μ_i control behavior of the interpolating function at the nodes [8]: when $0 < \mu_i \leq 1$ the interpolant is not differentiable at the i-th node; values of $\mu_i > 1$ assure that the interpolant is differentiable $\mu_i - 1$ times at the i-th node, but it has a flat spot there. In practice expression (7) is not convenient for numerical computations because it leads to $\frac{\infty}{\infty}$ in the neighborhood of the boundary of i-th geometrical object. These numerical problems are easily avoided by rewriting the weight functions in the equivalent but numerically stable form:

$$W_i\left(\mathbf{x}\right) = \frac{\prod\limits_{j=1;j\neq i}^{n} \omega_j^{\mu_j}\left(\mathbf{x}\right)}{\sum\limits_{k=1}^{n} \prod\limits_{j=1;j\neq k}^{n} \omega_j^{\mu_j}\left(\mathbf{x}\right)}. \tag{8}$$

The weighted inverse distance transfinite interpolation can be extended to interpolate the derivatives as it is shown in [24].