Geometrically Adaptive Numerical Integration

Brian Luft*
Vadim Shapiro†
University of Wisconsin-Madison

Igor Tsukanov‡
Florida International University

Abstract

Numerical integration over solid domains often requires geometric adaptation to the solid's boundary. Traditional approaches employ hierarchical adaptive space decomposition, where the integration cells intersecting the boundary are either included or discarded based on their position with respect to the boundary and/or statistical measures. These techniques are inadequate when accurate integration near the boundary is particularly important. In boundary value problems, for instance, a small error in the boundary cells can lead to a large error in the computed field distribution.

We propose a novel technique for exploiting the exact local geometry in boundary cells. A classification system similar to marching cubes is combined with a suitable parameterization of the boundary cell's geometry. We can then allocate integration points in boundary cells using the exact geometry instead of relying on statistical techniques. We show that the proposed geometrically adaptive integration technique yields greater accuracy with fewer integration points than previous techniques.

CR Categories: G.1.4 [Quadrature and Numerical Differentiation]: Adaptive and iterative quadrature; J.6 [Computer-Aided Engineering]: Computer-aided design (CAD) G.4 [Mathematical Software]: Algorithm design and analysis

Keywords: numerical integration, implicit function, quadrature, adaptive integration, meshfree analysis

1 Introduction

Many problems in physics and engineering require computation of integrals. Computation of surface areas and mass properties, such as weight, center of gravity, and moments of inertia, have been studied widely in solid modeling [Lee and Requicha 1982b; Lee and Requicha 1982a; Sarraga 1982; Cattani and Paoluzzi 1990; Gonzalez-Ochoa et al. 1998]. The important feature of all such computations is that they deal with the integration of *a priori known* functions over solid domains and/or their boundaries. Other physical quantities, such as work, energy, stiffness, and their sensitivities may involve integrating arbitrary *a priori unknown* functions — that is, functions constructed at run time — over solid domains. Engineering analysis applications rely on such integration for assembling response matrices and for computing integral properties of the physical fields being modeled. Most realistic shapes in engineering are too complex for symbolic integration of such functions, and so numerical methods for integration have to be used.

*e-mail: brianluft@mac.com
†e-mail:vshapiro@engr.wisc.edu
‡e-mail:igor.tsukanov@fiu.edu

In contrast to symbolic methods, numerical integration gives an approximate value of the integral. Volume integration in three-dimensional geometric domains can be performed by using *lattice rules* — sequential application of one-dimensional integration rules [Press et al. 1992] which can be reduced to computation of a weighted sum of the integrand's values computed at the integration nodes:

$$\iiint_{\Omega} f d\Omega = \sum_{i=1}^{n} w_i \sum_{j=1}^{n} w_j \sum_{k=1}^{n} w_k f(x_i, y_j, z_k). \qquad (1)$$

The location and number of integration nodes (x_i, y_j, z_k) are based on assumption that the integrand can be sufficiently accurately approximated by a polynomial of some degree. Thus, the nature of the integration error is multifactorial. It depends on the integrand f, geometry Ω, as well as the type and the order of the integration rule. In addition, the integration error includes an unavoidable roundoff error because numerical integration algorithms operate on floating-point numbers. For many applications, integration accuracy is critical. For example, in finite element analysis, integration error may lead to ill-conditioned elements of an algebraic system and propagate further into the solution of an engineering analysis problem. In order to keep the integration error under control and guarantee the desired accuracy, integration algorithms have to adapt to both the integrand and the geometry.

When the geometric domain Ω is a regular n-dimensional interval (a "box"), adaptivity to the integrand is well understood and can be achieved by choosing an appropriate integration rule and number of integration nodes. If *a priori* information about the integrand is unavailable, integration rules that allow *a posteriori* error estimation [Berntsen et al. 1991; Press et al. 1992; Kronrod 1965; Joe and Sloan 1993] can be used. If the integration error exceeds the specified value, the integration is performed using a increased number of integration nodes. However, increasing the order of the integration rule is not always a good strategy, especially if the integrand has singularities inside the integration domain. Alternatively, divide-and-conquer approach can be applied: if desired accuracy is not achieved for the given maximal number of integration points, the interval is subdivided and the integration procedure is recursively applied to its parts [Piessens et al. 1983; Cools and Maerten 1998; Berntsen et al. 1991].

When the integration domain Ω is not an n-dimensional box, allocation of integration nodes is no longer straightforward and is the cause for additional errors. Even though these geometry-induced errors can can be comparable or even sometimes greater than the errors caused by the approximate integrand, adaptation to geometry of the integration domain is not widely described in the literature. Among the implementations supporting multiple integration, the ability to use complex integration domains is quite uncommon. Integration domains are specified by an axis-aligned bounding box, requiring complex geometries to be specified using a Heaviside function in the integrand (we will discuss this approach in the Section 2). This reduces problem of geometric adaptivity to the adaptivity to the integrand. For comparison between algorithms, we compute the volume of two unit spheres side-by-side. The number of integration nodes used is varied to see the effect on the relative error in the integral value. We compare the geometrically adap-

tive algorithm, which is proposed in this paper and described in detail in Section 3, against four standard integration packages. The *DCUHRE* program described in [Berntsen et al. 1991] uses an 11-degree integration rule, and error approximation to determine its pattern of subdivision. The *RADMUL* program from the CERN Program Library is a modified version of the algorithm described in [Genz and Malik 1980], which is used by Mathematica's *NIntegrate* function. It uses a 7-degree integration rule and an adaptive subdivision strategy. Matlab's *QUADL* function uses adaptive Gauss-Lobatto quadrature as described in [Gander and Gautschi 2000]. Finally, the *GAULEG* procedure defined in [Press et al. 1992] implements nonadaptive integration over a simple lattice of Gauss-Legendre points. Figure 1 shows that the geometrically adaptive strategy presented in this paper offers improved accuracy for a given number of integration nodes than preexisting adaptive integration codes.

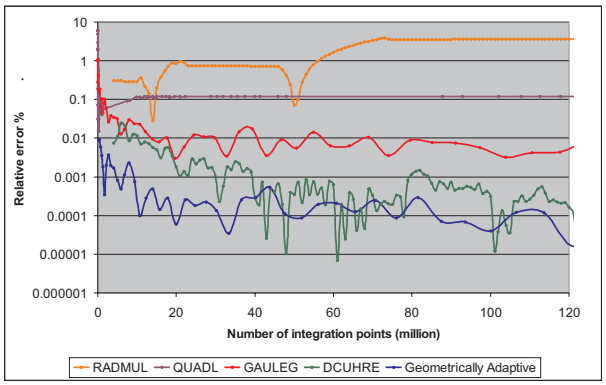


Figure 1: *Accuracy comparison between existing implementations of volume integration and the proposed geometrically adaptive integration technique*

This paper fills the gap by focusing entirely on geometrically adaptive integration. In particular, we explain inadequacies of earlier approaches and propose a computationally efficient method for three-dimensional geometrically adaptive volume integration that can be implemented in any system supporting standard queries. Our approach uses the original geometry of the integration domain represented by a characteristic function. Using this general description of the geometry, we can perform volume integration over a variety of geometric representations including Boundary Representation, CSG, meshed and polygonal geometric models.

Outline The rest of the paper is organized as follows: Section 2 reviews current techniques for handling the geometry of the integration domain and explains their weaknesses. Section 3 describes the proposed geometrically adaptive integration technique in detail. Some implementation issues are discussed in Section 4. In particular, we explain how our integration approach can be used to integrate over CAD geometric models as well as different integration modes which can be used for multiple integration over the same geometric model. Numerical examples in Section 5 demonstrate computational properties of the proposed integration technique. Section 6 concludes the paper and explains how the presented geometrically adaptive integration can be improved.

2 Background and related work

When the geometric domain Ω is irregularly shaped, careless or improper allocation of integration nodes may result in substantial accuracy degradation. A general principle informally suggested in [Press et al. 1992] and illustrated in Figure 2 is aimed to guarantee that all integration nodes are located inside Ω by scaling the one-dimensional lattice rules to fit Ω. However, it is not clear how to apply this principle in practice, or how to estimate and to control the resulting integration errors.

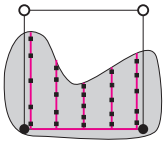


Figure 2: *One-dimensional lattice rules are applied along the vertical line segments that are themselves are allocated using the lattice rule in the horizontal direction.*

In order to achieve the desired integration accuracy, some form of geometry adaptation has to be employed. There are two well known ways to adapt to the geometry of the integration domain: (1) use hierarchical space decomposition methods; and (2) allocate integration points within an *extended* domain and use known integrand-adaptive techniques to adapt to the geometry.

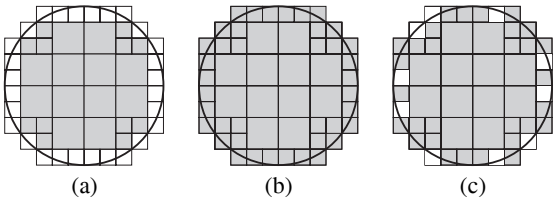


Figure 3: *Integration in boundary cells: (a) all boundary cells are excluded from the integration; (b) all boundary cells are used for the integration; (c) boundary cells are randomly excluded from the integration.*

The first approach — using hierarchical space decompositions such as quad/octtrees — has been used widely in solid modeling for computing mass properties [Lee and Requicha 1982a], as well as for computing integrals in computational mechanics [Klaas and Shephard 2000; Laguardia et al. 2005]. The simplicity and good computational properties made these geometric representations attractive for a variety of applications. Octree representation can also be viewed as a multiresolution representation where the level of geometric details can be easily controlled by the size of the smallest cell or by a subdivision level. With respect to the given geometric domain, octree cells fall into two groups: internal cells completely situated inside the geometric domain, and boundary cells which enclose portions of the domain's boundary. The simple geometric shape of the internal cells allows direct application of the lattice rules. The integration error in internal cells is completely determined by the integrand, and by the type and order of the numerical integration rule used to compute the value of an integral. In contrast, integration in boundary cells introduces additional numerical errors which are caused by the discrepancy of the octree with the geometry of the integration domain. This makes the volume integration in the boundary cells challenging.

Recursive subdivision of the boundary cells may be used to adapt to the given geometry domain Ω, but sooner or later the integrals over the boundary cells must be computed. There are several strategies for deciding on how to treat the boundary cells [Sarraga 1982]: (1) include all boundary cells and treat them as internal cells, (2) exclude boundary cells from the integration, and (3) randomly include and exclude boundary cells as shown in Figure 3. It is well known that the first two approaches suffer from low integration accuracy

(Figure 4) because complete inclusion or exclusion of the boundary cells introduces significant geometric error. The third approach provides better accuracy by balancing inclusion and exclusion of the boundary cells[Sarraga 1982]. Other criteria can be used to decide whether a boundary cell should be included or excluded. For example, if the integration domain occupies more than a half of a boundary cell, it is used for integration; otherwise the cell is ignored. A serious potential problem with all such approaches is that the integrand may not be defined in the boundary cells outside of the domain Ω. Furthermore, even if the balance between included and excluded cells is well maintained, the integration accuracy still remains relatively low. And last, but not least, because geometric adaptivity is achieved by subdivision of all boundary cells, this approach is computationally expensive. Each subsequent subdivision of the boundary cell leads to an eight-fold increase in the number of integration points allocated to this cell.

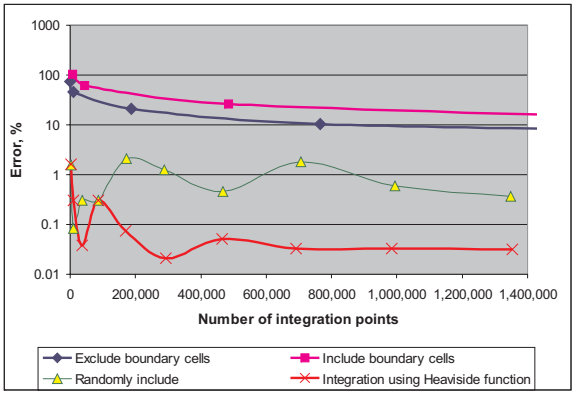


Figure 4: *Accuracy of volume computation of a unit sphere using different integration techniques.*

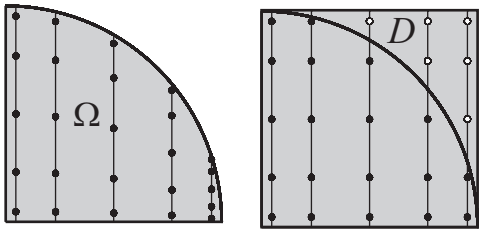


Figure 5: *(a) Integration domain. (b) Extended integration domain; white dots illustrate locations of the integration points outside of the geometric domain.*

Another approach to geometric adaptation transforms the integration of function f over any domain Ω into integration over an "extended" regular domain $D \supset \Omega$ of another function that is equal to f in Ω and vanishes on all other points of D. The approach is illustrated in Figure 5(b) and appears to be popular in the computational mechanics literature [Osher and Fedkiw 2003; Belytschko et al. 2003a; Kumar and Lee 2006]. It is based on the ability to construct for a given geometric domain Ω a characteristic function δ which is unity inside the domain and zero outside:

$$\delta(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \notin \Omega; \\ 1, & \mathbf{x} \in \Omega. \end{cases} \tag{2}$$

For solid models, such a function can be defined using a point membership test: if a geometric engine reports that the given point lies

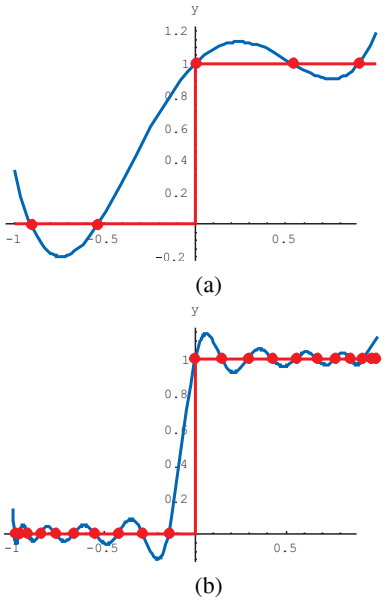


Figure 6: *Polynomial interpolation of a Heaviside function: (a) through 5 points; (b) through 21 points.*

inside the domain the value of a characteristic function is set to one. Otherwise it is zero. When the integration domain is described by an inequality $\omega(\mathbf{x}) \geq 0$, the characteristic function can be defined via a Heaviside function $\delta(\mathbf{x}) = H(\omega(\mathbf{x}))$. Using the characteristic function, a volumetric integral over a geometric domain Ω can be transformed into an equal integral over the extended domain D that encloses Ω (Figure 5):

$$\iiint_{\Omega} f(\mathbf{x})\, d\Omega = \iiint_{D} f(\mathbf{x})\, \delta(\mathbf{x})\, dV. \tag{3}$$

Integrals in the both sides of this expression are mathematically equivalent, and the problem of geometric adaptation is effectively transformed into the already solved problem of integrand adaptation.

But the two sides of the equation are different numerically. Consider the integrand in the right hand side of the expression (3). Because the characteristic function δ is discontinuous on the domain's boundary $\partial\Omega$, the integrand $f(\mathbf{x})\,\delta(\mathbf{x})$ inherits this discontinuity. If this function is numerically integrated using, for example, Gauss integration rules, accuracy of the result will be questionable. It is well known that numerical integration rules are derived based on the assumption that the integrand is a polynomial of some degree [Press et al. 1992]. For instance, n point Gauss integration rule gives accurate results for all polynomials up to degree $2n-1$. If the same integration rule is applied to a non-polynomial function, it gives a value of the integral of a polynomial approximation to the integrand. Thus, the accuracy of the approximation of the integrand by polynomials determines the integration accuracy. It is also well known that polynomial approximations of discontinuous functions tend to oscillate in the neighborhood of the discontinuity. For example, plots in Figure 6 illustrate polynomial approximation of a Heaviside function by a polynomials of $4th$ (Figure 6(a)) and $20th$ (Figure 6(b)) degree on the segment $[-1, 1]$. If we raise the degree of an approximating polynomial, the amplitude of oscillations raises as well. This implies that application of Gauss integration rules to discontinuous integrand functions cannot provide sufficiently accurate results. Several authors [Osher and Fed-

kiw 2003; Kumar and Lee 2006; Belytschko et al. 2003b; Wang and Wang 2006] suggested to construct a characteristic function δ using smoothed step functions instead of the Heaviside function. Smoothing reduces oscillations of the approximating polynomials, but it also affects the integrand in the boundary's neighborhood which, in turn, leads to accuracy degradation. Integration accuracy can be improved by the divide-and-conquer approach which we mentioned earlier in this Section. But, unfortunately, numerous boundary cell subdivisions make this geometrically adaptive approach computationally very expensive [Kumar and Lee 2006].

Several other approaches to geometrically adaptive integration of two-dimensional domains have been reported [Stroubolis et al. 2001; Höllig 2003; Rvachev et al. 1994; Tsukanov and Shapiro 2002]. In [Höllig 2003], the author proposed to impose a Cartesian grid over a two-dimensional geometric domain such that integration in boundary cells is performed over smoothly deformed rectangles. The approach described in [Rvachev et al. 1994] proposes new integration rules for boundary cells classified by the standard marching cubes algorithm, and a geometrically adaptive extension of this approach was proposed in [Tsukanov and Shapiro 2002]. In this paper we formulate key principles for geometrically adaptive integration and generalize approaches described in [Rvachev et al. 1994; Tsukanov and Shapiro 2002] to 3D. We also describe software implementation and present experimental evidence of superior performance of the proposed integration approach.

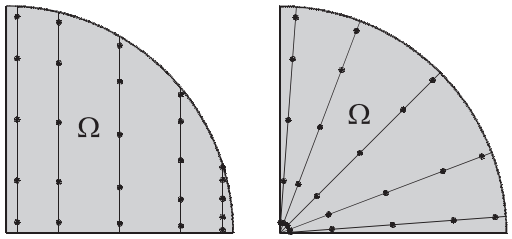


Figure 7: *Placement of the integration nodes using (a) Cartesian and (b) polar coordinates*

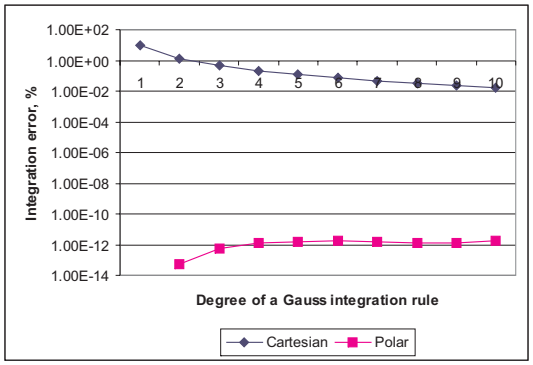


Figure 8: *Integration error of numerical evaluations of the integrals (5) and (6)*

3 Adaptive parameterization

3.1 Importance of Coordinate System

In this paper we propose a new geometrically adaptive integration technique which can be considered an adaptive parameterization of Ω. It is built on the combination of two ideas: (1) convenient space parameterization of the boundary integration cells and (2) hierarchical space decomposition. What does parameterization have to do with geometric adaptation? It turns out that using *different* coordinate systems for computing the *same* integral numerically we obtain results with different accuracy. Let us illustrate this fact on a simple example. The area of a quarter of a circle shown in Figure 7 can be defined as an integral of a unity function over a geometric domain Ω:

$$I = \iint_{\Omega} 1 d\Omega. \tag{4}$$

This integral can be evaluated analytically in Cartesian coordinates

$$I = \iint_{\Omega} 1 d\Omega = \int_0^1 \int_0^{\sqrt{1-x^2}} 1 dy dx = \int_0^1 \sqrt{1-x^2} dx = \frac{\pi}{4} \tag{5}$$

and in polar coordinates:

$$I = \iint_{\Omega} 1 d\Omega = \int_0^{\frac{\pi}{2}} \int_0^1 \rho d\rho d\phi = \int_0^{\frac{\pi}{2}} \frac{1}{2} d\phi = \frac{\pi}{4}. \tag{6}$$

In both instances we obtained the same result, which is not surprising. However, numerical evaluation of the integrals (5) and (6) using, for example, Gauss integration rules, gives different results. Graph in Figure 8 illustrates a sufficiently high integration error when integral (4) is computed in Cartesian coordinates (Figure 7(a)). The integration error cannot be reduced by increasing the number of integration points. In contrast, numerical evaluation of the integral (4) in polar coordinates results in a negligibly small integration error. This difference in integration accuracy can be easily explained. Numerical evaluation of the integral (4) in Cartesian coordinates is reduced to computation of the integral $\int_0^1 \sqrt{1-x^2} dx$. Singularity of the integrand at $x = 1$ determines significant integration error when Gauss integration rules are used. When the integral (4) is evaluated in polar coordinates, the singularity in the integrand disappears and numerical integration produces result with almost no error (Figure 8). This example illustrates that the integration error depends on the space parameterization of the underlying integration domain. Suitable parameterization can eliminate singularities in the integrand and, therefore, reduce the integration error.

Finding a good parameterization for an arbitrary shaped geometric domain Ω appears to be a difficult problem, but the application of hierarchical space decomposition reduces the parameterization problem to parameterization of only those cells that intersect the boundary of Ω. Interior cells are integrated by directly applying the usual lattice rules to allocate nodes throughout the cell. Cells that intersect the boundary of the integration domain require more work and attention, because numerical integration in these cells introduces a "geometric" part of the integration error. Now, our goal is to find for each boundary cell an appropriate space parameterization that reduces this error.

3.2 Geometric interpretation of lattice rules

Let c be a boundary cell, and $X = c \cap \Omega$ is the portion of Ω contained inside the boundary cell c. Because all lattice rules stem from repeated application of the one-dimensional integration over a connected closed interval, they apply only if X is homeomorphic to an n-dimensional box (or equivalently, an n-ball). We shall assume

this to be the case, but we will discuss how this requirement may be enforced in practice in Section 3.4.

As our example suggests, the choice of a proper space parameterization of a boundary cell depends on how the cell intersects the domain's boundary $\partial\Omega$. More specifically, the existence of singularity may be determined by examining the geometry (e.g., the angle) of intersection between ∂c and $\partial\Omega$. This approach appears to be impractical when high accuracy of integration demands processing of large number of boundary cells. Furthermore, we are more interested in the behavior of the numerical approximation, than in the details of the original geometry. The repeated application of one-dimensional application of the lattice rule in expression (1) has a simple geometric interpretation as is evident from Figure 7. If $x_i, i = 1, 2, 3$ are the coordinates used to parameterize X, then for any fixed values of x_1 and x_2, the integration procedure reduces to allocating integration nodes on one-dimensional interval spanned by the values of x_3. Each of this intervals correspond to a line segment, and every choice of coordinate system for X implicitly parameterizes the integral over X in terms of one-dimensional lattice rules along the corresponding set of line segments. Thus, in Figure 7, the parameterizing the quarter of a circle in Cartesian system implies that the one-dimensional lattice rules are applied on a set of vertical line segments, while polar coordinate system implies that the lattice rules are applied to a set of rays emanating from the origin. In case of the Cartesian system, the rays become shorter and shorter, eventually becoming tangent to the boundary of X. By contrast, in the polar coordinate system, all rays intersect the boundary of X transversally.

Based on the above observations, we postulate that a good choice of coordinate system will allocate the integration rays to (1) maximize the number of rays that do not intersect $\partial\Omega$; and (2) increase the likelyhood that all rays intersect $\partial\Omega$ transversally and only once. These principles are clearly heuristic and do not provide any theoretical guarantees; but our implementation and experiments indicate that the resulting practical classification and parameterization of the boundary cells yields a geometrically adaptive integration that is superior to previous approaches.

3.3 Classification of boundary cells

The boundary integration cell $X = c \cap \Omega$ can be parameterized using Cartesian, cylindrical or spherical coordinate systems. The usual Cartesian lattice rules are applied when $c \subset \Omega$; in this case, X is the interior cell, none of the integration rays intersect $\partial\Omega$ and no geometric computations are needed. Cartesian coordinates can be also used to integrate over the cells X that are "mostly inside" the domain Ω. More precisely, if at least one of the faces Y of X is fully contained in Ω, any choice of the coordinate system can be used to parameterize face Y, since none of the rays lying in this face intersect $\partial\Omega$. Choosing the Cartesian coordinate system for all three variables, allocation of the integration nodes starts with imposing an initial $N \times N$ grid on the interior face according to the chosen Gauss integration rule. The nodes of this grid are the starting points for the rays which extend to $\partial\Omega$. The same lattice rule is used to allocate integration nodes along each ray. See Figure 9.

If no faces of X are interior to Ω, we choose a different type of coordinate system that would allow us to use all edges of X that are interior to Ω, while increasing the likelyhood that the rest of integration rays intersect $\partial\Omega$ transversally. If there is only one such internal edge (Figure 10), it is convenient to parameterize the space and perform the integration using cylindrical coordinates. As previously, the parameterizing procedure is based on the same ray-boundary intersection principle. First, N starting points are allocated between the two interior vertices, according to the Gauss integration rule. Each of these points defines a section of X that is parameterized using the polar coordinates, analogously to parameterization of the quarter of a circle. In each section, from each of these points, N coplanar rays are shot until intersection with $\partial\Omega$. The rays are arranged radially around the initial edge, according to the chosen Gauss rule within the 90° span between the two adjacent cell faces.

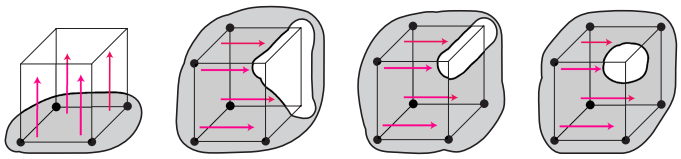


Figure 9: *Integration using Cartesian coordinates: most cell's vertices lie inside the integration domain*

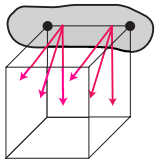


Figure 10: *Integration using cylindrical coordinates*

If the only one vertex of a cell is inside the domain, the boundary cell X can be parameterized using a spherical coordinates. The vertex becomes the sole starting point for $N \times N$ rays. As the cylindrical parameterization used Gauss nodes to determine θ, the spherical technique uses them to create a "grid" of ϕ and θ ray angles. Each ray is intersected with the boundary and used to place the integration nodes between the cell's vertex and the intersection point. Spherical coordinates are also used when one corner is inside the domain as well as the three corners adjacent to it as it is shown in Figure 11.

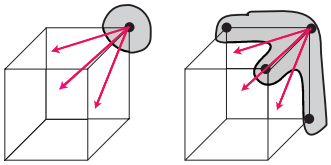


Figure 11: *Integration using spherical coordinates*

The above rules are proposed in the spirit of postulated principles aimed to increase the accuracy of numerical integration. While our experiments (see below) support the expected improved performance, the proposed rules provide no theoretical guarantees and are not unique. In particular, to assure that every ray intersect the boundary ∂X only once, X must be star-shaped in the case of the spherical coordinate system, monotone in the case of Cartesian coordinate system, and both in the case of cylindrical coordinate system. There is no reason to expect that this is always the case; however multiple ray intersections are readily detected and in this case X is subdivided again.

3.4 Marching cubes and small features

A reader may have noticed that the classification of the boundary cells closely resembles the classification of the boundary cells in

the classical marching cube algorithm [Lorensen and Cline 1987b]. Under suitable assumption about absence of small features, all boundary cells are classified by checking which of the eight vertices of the cell are inside the integration domain and which are outside. With eight vertices we can obtain 256 different intersection cases which can be reduced by symmetry transformations to only 8 basic cases [Lorensen and Cline 1987b]. It is also well known that some of the cases are ambiguous and, therefore, cannot be used for volume integration. Geometry of the region of the integration domain that lies inside such boundary cell cannot be determined from just the point membership of the eight corners. For instance, in Figure 12 it is unclear whether the geometry in this cell is a contiguous strip between the two corners that are interior to the geometry, or whether the geometry is two disconnected islands. These and other ambiguities have been studied in literature and can be avoided when a number of sampling and topological conditions are satisfied (for example, see [Varadhan et al. 2004]). When the conditions are not satisfied, ambiguous cells must be detected and subdivided to simplify the geometry in each cell, until the ambiguities are resolved *or* the size of cell is consistent with the acceptable errors in integration.

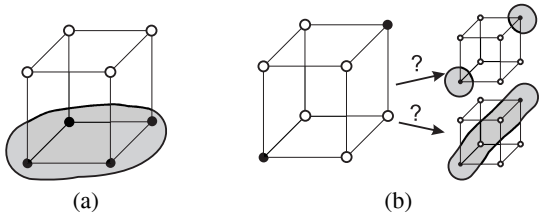


Figure 12: *Boundary integration cells: (a) unambiguous; (b) ambiguous*

Testing values of a characteristic function at cell's vertices is a relatively coarse geometry identification method. If a geometric feature or a void is smaller than a cell, it may not be discovered by this method. Figure 13(a) illustrates this situation: the void is completely situated inside the cell. Since all vertices of a cell are inside the geometric domain, this void is missed during the allocation of the integration nodes, and, as a result, the integration accuracy will deteriorate. There are several ways to detect small geometric features and voids.

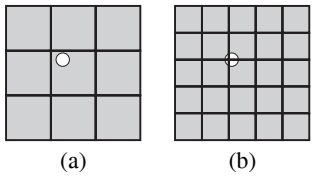


Figure 13: *Small feature detection by increasing grid resolution: (a) missed feature; (b) detected feature*

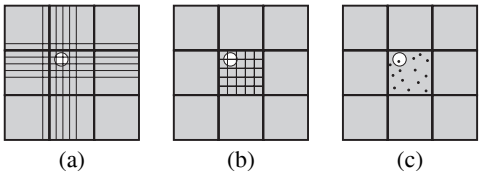


Figure 14: *Small feature detection by (a) imposing a non-uniform grid; (b) imposing a uniform grid within a cell; (c) sampling random points*

If the user knows that small features exist in the domain geometry, an initial grid with tighter spacing (Figure 13(b)) can be provided to ensure a cell corner intersects the feature. This is a simple way to globally ensure that features down to a given size are detected. However, because the increased grid resolution applies everywhere in the domain, including regions where it is not needed, it can unnecessarily drive up the computational cost. This approach can be optimized if an additional information about location of small geometric features is available. In this case, as it is illustrated by Figures 14(a) and (b), a non-uniform initial grid can be used, or a denser grid is imposed within a cell.

Checking the values of a characteristic function at randomly distributed points offers another alternative. When totally interior or totally exterior cells are encountered, the characteristic function is computed at some randomly-placed points, as shown in Figure 14(c), to ensure the cell does not contain any undetected voids or features. If a such region is detected, the cell is subdivided and integration is retried on each of the subcells. The user can specify the number of random points to check in each cell, and increasing that number increases the likelihood of catching small features. Random points are a good safeguard against missing small features, but sampling the characteristic function for every random point and for every interior or exterior cell can be computationally expensive.

When the integration domain Ω is provided by a solid modeling kernel such as Parasolid, the engine's native cube-boundary intersection function can be used to determine whether a cell c is truly completely interior or exterior. If cell $X = c \cap \Omega$ contains any interior voids, it is subdivided.

4 Implementation issues

We have implemented the described geometrically adaptive integration technique in a C++ class called VolumeIntegrator. The core of the integration procedure is the set of C++ functions which parameterize a given boundary cell. Each type of parameterization — Cartesian, cylindrical, and spherical — is implemented by a separate, generalized function which uses its arguments to tailor the parameterization to the specific boundary intersection case. For instance, the spherical parameterization is used when only one of the eight vertices is inside the geometry. To handle each of these cases, the parameterization function takes as input the vertex which is inside, and the range of θ and ϕ values which will ensure that nodes will not be placed outside of the cell. The other two parameterizations take similar ranges as input. Because the inputs to these functions differ for each boundary intersection case, it is convenient to store the information in a table. The table has 256 entries, one for each possible intersection case. Each entry is a structure which specifies which parameterization to use (or if the cell should be subdivided) and includes the extra input that the parameterization functions require.

VolumeIntegrator class provides user-friendly interface to the integrand and functions defining the integration domain. The C callback function mechanism is used to allow easy integration of any geometry representation. The user provides the point membership test as a pointer to a callback function which takes a point in space (X, Y, Z) and returns whether that point is interior or exterior. Optional cube-geometry and ray-boundary intersection functions are similarly implemented as callback functions. To use a geometric engine like Parasolid, the user needs only to implement these callback functions as wrappers around the engine's API functions.

Also provided to the integration procedure is the bounding box for the geometry. This bounding box is not the boundary of the integration domain, it contains the domain which is implicitly defined by the point membership function. The number of integration nodes per dimension per cell is provided; if the user requests 5 nodes per

dimension, then each integration cell will contain $5^3 = 125$ nodes. The initial grid can be specified as either a uniform $n_x \times n_y \times n_z$ lattice or as a list of nonuniform lattice intersection locations.

Our implementation of VolumeIntegrator class provides two modes of integration: on-the-fly and on-disk. On-the-fly integration performs node allocation and integration in one pass. As leaf cells are encountered, integration nodes are allocated, the integrand is sampled, and node weights are computed. After the computation of a cell's contribution to the integral result, the cell can be thrown away. In this way, only one cell is held in memory at a time, and the octree is implied by recursive calls to the integration procedure. Therefore, both memory and disk requirements for the on-the-fly algorithm are constant, regardless of the number of integration cells and the maximum level of subdivision.

On-disk integration mode separates node allocation from integration. The first phase generates the octree and places all integration nodes, storing the results to a file on a disk. With this precomputed octree, any number of integrations can be performed by providing an integrand and running just the second phase of the integration. When performing a series of integrations with differing integrands over the same domain, on-disk integration offers substantial savings in computation time, because the octree and node allocations need to be performed only once. The on-disk integration procedure has constant memory requirements, but disk storage requirements grow with the number of leaf cells in the octree.

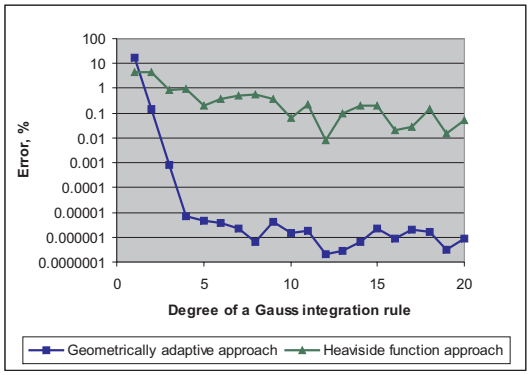
The VolumeIntegrator class provides convenient interface that allows fast integration in time-varying geometric domains. Some applications requiring a series of integrations also require slight, local changes to the shape of the integration domain. Instead of recomputing the entire octree and node placements with each step, computation time can be saved by taking advantage of the knowledge that most of the domain remains the same between steps. Between successive integrations, the user can provide a bounding box which contains the changes to the integration domain. The on-disk octree is traversed and any integration cells intersecting the changes bounding box are thrown away. Integration node placements for the thrown away cells are recomputed, while the rest of the integration domain retains the prior, untouched node placement. In this way, the octree can be "patched" to include domain shape changes without fully recomputing the octree.

5 Numerical examples

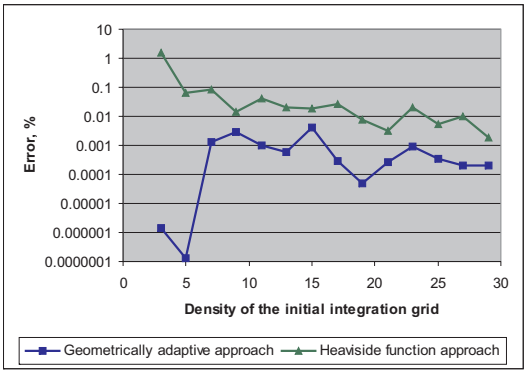
To verify the accuracy of our geometrically adaptive integration algorithm we perform several numerical experiments.

Volume computation of a unit sphere This experiment determines the accuracy of numerical integration of a unity function over a unit sphere. Choosing a unity function as an integrand, we eliminate numerical errors associated with the integrand. This means that the integration error is completely determined by the geometry of the integration domain. In addition, integrating a unity function simplifies error computations, because the numerical value of the integral gives a volume of the underlying geometric domain, which has an exact value for simple geometric domains such as a unit sphere.

Figure 15(a) illustrates dependence of the relative integration error on the degree of a Gauss integration rule. To perform the integration we imposed a uniform Cartesian grid $3 \times 3 \times 3$ over a bounding box that contains a unit sphere. The plot in Figure 15(a) demonstrates exponential convergence of our geometrically adaptive integration algorithm. It provides much more accurate results than the integration approaches that use a Heaviside function including those used



(a)

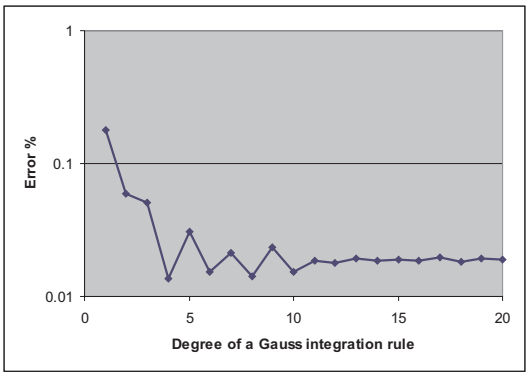


(b)

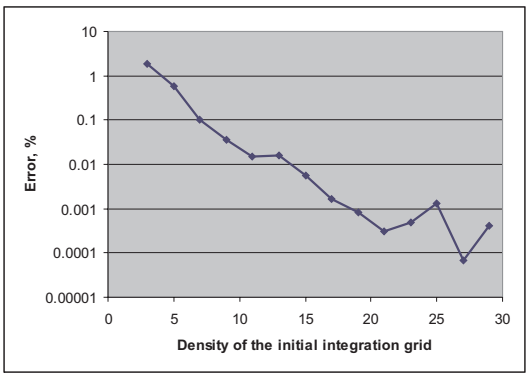
Figure 15: *Dependence of the integration error over a unit sphere on (a) degree of a Gauss integration rule (3×3 initial grid); (b) density of the initial integration grid (Gauss rule of 10th degree)*

by the commercial packages compared in Section 1.

Dependence of the integration error on the density of the initial Cartesian grid is shown in Figure 15(b). The computations were performed using a Gauss rule of $10th$ degree. This plot and the plot in Figure 1 illustrate better accuracy of a geometrically adaptive integration in comparison with a Heaviside function approach. Convergence rate of a geometrically adaptive integration with increasing density of the initial grid is slower than with increasing the order of the integration rule. Increasing density of the initial integration grid changes the type of space parameterization in the integration cells, which, in turn, causes some oscillations of the integration error. However, increasing density of the initial integration grid can be useful if the integration domain contains small geometric features, as illustrated by the next experiment.



(a)



(b)

Figure 16: *Integration error over a cube with five random spherical voids: (a) using* $10 \times 10 \times 10$ *initial integration grid; (b) using Gauss integration rule of 10th degree*

Volume of a cube with randomly distributed spherical cavities This numerical experiment illustrates automatic detection of small geometric features in the integration domain. We choose integration domain to be a cube defined by its extreme vertices $(-1, -1, -1)$ and $(1, 1, 1)$ with five spherical voids of random radii and with centers located at random points (see Figure 18). The size and location of these voids are chosen in a such way that they do not intersect with an initial $10 \times 10 \times 10$ uniform integration grid. The plot in Figure 16(a) illustrates dependence of the relative integration error on the degree of Gauss integration rule. For degrees greater 10, the integration error stabilizes at the value of 0.019%. This suggests that despite using 10 random points to find small voids, some of them have been missed. As we discussed earlier, increasing the density of the initial grid enhances the chances that small geometric features will be detected. Plot in Figure 16(b) reveals that in the presence of small geometric features density increase of the integration grid may substantially improve the integration accuracy.

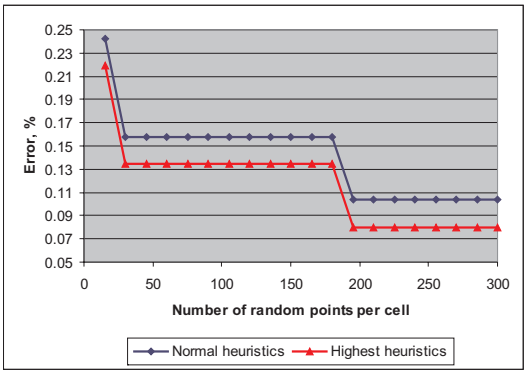


Figure 17: *Detection of small geometric features using randomly sampled points. Drastic changes in the integration error indicate detection of small voids*

X_c	Y_c	Z_c	R
-0.396954	0.120579	0.225074	0.103729
-0.066866	0.117954	-0.663381	0.219211
0.350078	-0.823847	0.305155	0.164589
0.300638	0.673696	0.084994	0.252837
-0.639149	0.792718	0.457930	0.133271

Figure 18: *Location and radii of five randomly distributed spherical voids*

Sampling of random points is another way of detecting small and irregular geometric features. Figure 17 shows dependence of an integration error on the number of random points sampled. The geometric domain is a cube defined by its extreme vertices $(-1, -1, -1)$ and $(1, 1, 1)$ with three spherical voids (see Figure 19). Numerical integration was performed without imposing any initial grid. If no random points were sampled all these voids will be missed because they do not intersect with the boundary of the integration cell. Plots in in Figure 17 clearly demonstrate detection of all three voids by sampling random points only inside the cell, and, in addition to, by sampling random points on the faces and edges of the integration cells. As we described earlier, when a small geometric feature is found, the cell is hierarchically subdivided and the same procedure is applied to all of its children cells.

Integration over solid model This numerical experiment compares the integration accuracy for two different representations of the same geometric domain (Figure 20): (1) by a characteristic function and (2) by Parasolid geometric engine. Parasolid, in ad-

X_c	Y_c	Z_c	R
-0.726554	-0.764214	0.857295	0.05
-0.244179	0.102634	0.732963	0.1
-0.321635	-0.283609	-0.363933	0.4

Figure 19: *Location and radii of three randomly distributed spherical void*

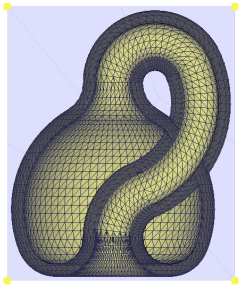
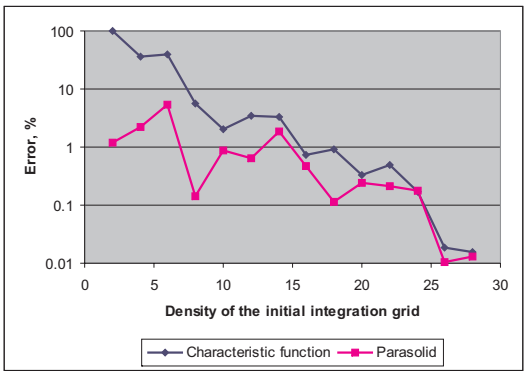
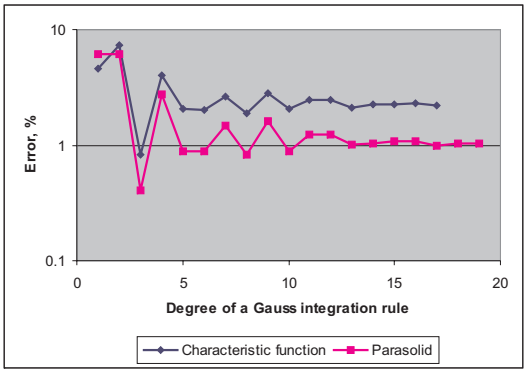


Figure 20: *CAD geometric model*



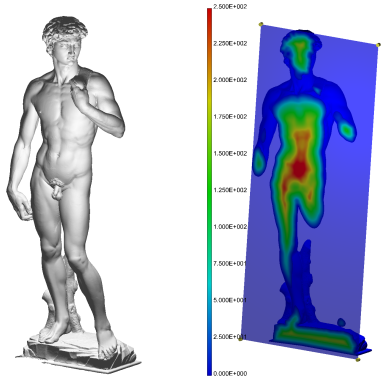
(a)



(b)

Figure 21: *Comparison of the integration accuracy for two representations of the same geometric model. (a) Dependence of the integration error on the density of the initial grid (Gauss integration rule of 5th degree); (b) dependence of the integration error on the degree of a Gauss rule* ($10 \times 10 \times 10$ *initial integration grid)*

dition to the point membership test, provides all geometric tools to detect small and irregular features without sampling random points. Plots in Figures 21 show that better accuracy can be achieved if the geometric model is represented using Parasolid. As we explained earlier, our integration approach assumes that integration rays intersect the domain's boundary only once. Using a characteristic function alone to represent the geometric domain, it is impossible to detect multiple intersections of the integration ray with the domain's boundary. This results in higher integration error. Plot in Figure 21(a) illustrates that increasing the density of the initial integration grid improves the integration accuracy for the domain represented by a characteristic function.



(a) (b)

Figure 22: *(a) Acquired geometric model of the Michelangelo's David statue; (b) signed approximate distance to the boundary of a geometric model shown in Figure 22(a)*

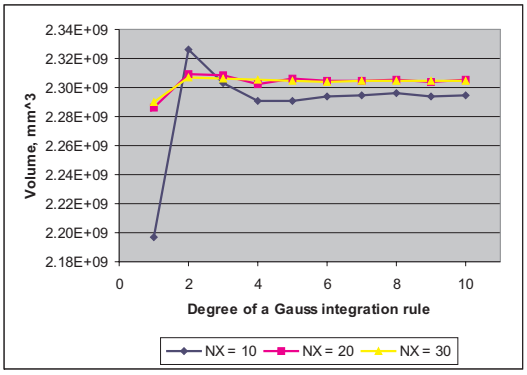


Figure 23: *Convergence of the volume computation of the Michelangelo's David statue*

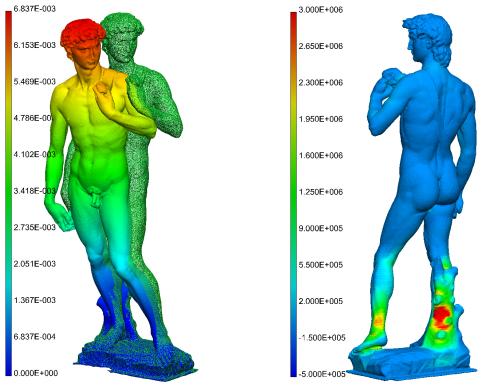
Integration over acquired geometric data This numerical experiment studies convergence of the volume integration over complex geometric object, such as the Michelangelo's David statue (Figure 22(a)). The geometry of the integration domain is described by a signed approximate distance function ω (Figure 22(b)) that was constructed from the originally acquired geometric data [Bracci and et. al. 2004]. To simplify reading, plot in Figure 22(b) shows only the positive portion of ω. The sign of ω helps to distinguish internal points from the outer ones ω is positive inside the geometric domain and negative outside. Using the sign of ω we define a characteristic

function for the statue as follows:

$$\delta = \begin{cases} 1, & \omega \geq 0; \\ 0, & \omega < 0. \end{cases} \tag{7}$$

Figure 23 illustrates numerical convergence of the volume computations via integrating a unity function over a geometric domain described by the characteristic function (7). The volume integrals were computed using different initial grids and degrees of a Gauss integration rule. The plots exhibit almost identical results for grids $20 \times 20 \times 20$ and $30 \times 30 \times 30$. The plots reveal insignificant variation of the computed integrals for degrees of a Gauss integration rule greater than 4.

6 Conclusions



(a) (b)

Figure 24: *Meshfree stress analysis of the Michelangelo's David statue for tilted position of the statue ($\varphi_x = -3^\circ$, $\varphi_y = 3^\circ$): (a) displacement field (mm); (b) first principal stress (Pa)*

In this paper we described a geometrically adaptive integration approach that is faster for the same accuracy or, in different words, provides better numerical accuracy for the same computational cost. This was achieved by using the provided geometric information directly, and by imposing in each integration cell a proper coordinate system that simplifies parameterization of the domain's boundary within that cell. In order to achieve the required accuracy, our geometrically adaptive integration approach does not necessarily require hierarchical subdivision of boundary cells — only those cells that have difficulty to impose a suitable space parameterization or contain small or irregular geometric features are subdivided. This substantially reduces the computational cost of the proposed approach in comparison to other volume integration techniques.

The presented geometrically adaptive integration approach requires minimal knowledge about geometry of the integration domain. It is based on a general representation of the integration domain by a characteristic function. Since most geometric representations support computation of a such function, our integration technique is applicable to a wide class of geometric representations. However, if an additional geometric information is available, it can be also used by the integration algorithm. Combination with a standard geometric engines, such as Parasolid, improves both accuracy and computational cost of volumetric integration by detecting small and irregular geometric features, and avoiding heuristic geometry checks at random locations. Besides computing values of a characteristic function, the proposed integration approach also requires computation of the intersections of rays with the domain's boundary. This geometric operation can be easily performed for simple geometric boundaries such as cylinders, spheres, conical sections and meshes. For parametric patches, however, the elevated computational cost of the ray intersection with a boundary is well justified by the high integration accuracy delivered by the presented geometrically adaptive integration technique.

Software implementation of our geometrically adaptive integration provides convenient interface to user-defined functions and can support 3D volumetric integration for a variety of applications. In this paper we already demonstrated integration over implicitly defined geometric domains. The proposed integration approach was developed to support 3D meshfree analysis [Freytag et al. 2006; Freytag et al. 2007]. All meshfree engineering analysis methods assemble the stiffness matrix by integrating shape functions and their derivatives over non-meshed geometric domains [Tsukanov and Shapiro 2002]. Accuracy of the integration determines stability and accuracy of the fields being modeled. Geometrical adaptivity makes it possible to perform field modeling in geometrically complex domains, such as, for example, statues and human bones without any simplifications of the geometry. Figure 24(a) presents the original (undeformed) and deformed statue of the Michelangelo's David. The first principal stress computed for tilted position of the statue ($\varphi_x = -3^\circ$, $\varphi_y = 3^\circ$) under its own weight (Figure 24(b)) are in a good agreement with the results obtained by the Finite Element Method [Bracci and et. al. 2004].

Acknowledgments

This research is supported in part by the National Science Foundation grants CMMI-0323514, CMMI-0322134, CMMI-0621116 and Intact Solutions, LLC. The authors would like to thank Professor Marc Levoy for providing original 3D scanned data of the David statue; and Professor Krishnan Suresh for many suggestions, encouragement and support of this research.

References

BELYTSCHKO, T., PARIMI, C., MOËS, N., SUKUMAR, N., AND USUI, S. 2003. Structured extended finite element methods for solids defined by implicit surfaces. *International Journal for Numerical Methods in Engineering 56*, 4 (January), 609–635.

BELYTSCHKO, T., XIAO, S., AND PARIMI, C. 2003. Topology optimization with implicit functions and regularization. *International Journal for Numerical Methods in Engineering 57*, 8, 1177–1196.

BERNTSEN, J., ESPELID, T., AND GENZ, A. 1991. An Adaptive Algorithm for the Approximate Calculation of Multiple Integrals. *ACM Transactions on Mathematical Software 17*, 4 (December), 437–451.

BLOOMENTHAL, J. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers.

BRACCI, S., AND ET. AL. 2004. *Exploring David: diagnostic tests and state of conservation*. Giunti Editore S.p.A., Florence-Milan.

CATTANI, C., AND PAOLUZZI, A. 1990. Symbolic analysis of linear polyhedra. *Engineering with Computers 6*, 1, 17–29.

COOLS, R., AND HAEGEMANS, A. 2003. Algorithm 824: CUBPACK: A Package for Automatic Cubature; Framework Description. *ACM Transactions on Mathematical Software 29*, 3 (September), 287–296.

COOLS, R., AND MAERTEN, B. 1998. A Hybrid Subdivision Strategy for Adaptive Integration Routines. *Journal of Universal Computer Science 4*, 5, 486–500.

COOLS, R., LAURIE, D., AND PLUYM, L. 1997. Algorithm 764: Cubpack++: A C++ Package for Automatic Two-Dimensional Cubature. *ACM Transactions on Mathematical Software 23*, 1 (March), 1–15.

FREYTAG, M., SHAPIRO, V., AND TSUKANOV, I. 2006. Field modeling with sampled distances. *Computer Aided Design 38*, 2, 87–100.

FREYTAG, M., SHAPIRO, V., AND TSUKANOV, I. 2007. Scan and solve: Acquiring the physics of artifacts. In *Proceedings of the 2007 ASME International Design Engineering Technical Conference*.

GANDER, W., AND GAUTSCHI, W. 2000. Adaptive quadrature—revisited. *BIT Numerical Mathematics 40*, 1, 84–101.

GENZ, A., AND COOLS, R. 1993. Algorithm 720: An Algorithm for Adaptive Cubature Over a Collection of 3-Dimensional Simplices. *ACM Transactions on Mathematical Software 19*, 3 (September), 320–332.

GENZ, A., AND COOLS, R. 2003. An Adaptive Numerical Cubature Algorithm for Simplices. *ACM Transactions on Mathematical Software 29*, 3 (September), 297–308.

GENZ, A., AND MALIK, A. 1980. Remarks on algorithm 006: An adaptive algorithm for numerical integration over an n-dimensional rectangular region. *Journal of Computational and Applied Mathematics 6*, 295–302.

GONZALEZ-OCHOA, C., MCCAMMON, S., AND PETERS, J. 1998. Computing Moments of Objects Enclosed by Piecewise Polynomial Surfaces. *ACM Transactions on Graphics 17*, 3, 143–157.

GUO, X., AND QIN, H. 2005. Real-time mesh-free deformation. *Computer Animation and Virtual Worlds 16*, 3-4 (July), 189–200.

HAMMERSLEY, J., AND HANDSCOMB, D. 1964. *Monte Carlo Methods*. Methuen, London.

HÖLLIG, K. 2003. *Finite Element Methods with B-Splines*. No. 26 in Frontiers in Applied Mathematics. SIAM.

JOE, S., AND SLOAN, I. 1993. Implementation of a Lattice Method for Numerical Multiple Integration. *ACM Transactions on Mathematical Software 19*, 4 (December), 523–545.

KALOS, M., AND WHITLOCK, P. 1986. *Monte Carlo Methods*. Wiley, New York.

KLAAS, O., AND SHEPHARD, M. 2000. Automatic generation of octree based three dimensional discretizations for partition of unity methods. *Computational Mechanics 25*, 2–3, 296–304.

KRONROD, A. S. 1965. *Nodes and Weights of Quadrature Formulas: Sixteen place tables*. Consultants Bureau, New York.

KUMAR, A., AND LEE, J. 2006. Step function representation of solid models and application to mesh free engineering analysis. *Journal of Mechanical Design(Transactions of the ASME) 128*, 1 (January), 46–56.

LAGUARDIA, J., CUETO, E., AND DOBLARE, M. 2005. A natural neighbor Galerkin method with quadtree structure. *International Journal for Numerical Methods in Engineering 63*, 6, 789–812.

LEE, Y. T., AND REQUICHA, A. A. G. 1982. Algorithms for computing the volume and other integral properties of solids. I. known methods and open issues. *Commun. ACM 25*, 9, 635–641.

LEE, Y. T., AND REQUICHA, A. A. G. 1982. Algorithms for computing the volume and other integral properties of solids. II. a family of algorithms based on representation conversion and cellular approximation. *Commun. ACM 25*, 9, 642–650.

LIEN, S., AND KAJIYA, J. T. 1984. Symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *IEEE Computer Graphics and Applications 4*, 10, 35–41.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching Cubes: A high resolution 3D surface construction algorithm. In *ACM SIGGRAPH Computer Graphics, Proceedings of the 14th annual conference on Computer graphics and interactive techniques SIGGRAPH '87*, ACM Press, New York, NY, USA, vol. 21, 163–169.

LORENSEN, W., AND CLINE, H. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics 21*, 4, 163–169.

OSHER, S., AND FEDKIW, R. 2003. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag.

PIESSENS, R., DEDONCKER KAPENGA, E., UEBERHUBER, C., AND KAHANER, D. 1983. *QUADPACK: A Subroutine Package for Automatic Integration*. Springer, Berlin; New York.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. *Numerical Recipes in C*, second ed. Cambridge University Press.

RVACHEV, V., SHEVCHENKO, A., AND VERETELNIK, V. 1994. Numerical integration software for projection and projection-grid methods. *Cybernetics and Systems Analysis 30*, 1 (January), 154–158.

SARRAGA, R. 1982. Computation of surface areas in GMSolid. *IEEE Computer Graphics and Applications 2*, 7, 65 – 70.

SHEFFER, A., AND ÜNGÖR, A. 2001. Efficient adaptive meshing of parametric models. *Journal of Computing and Information Science in Engineering 1*, 4, 366–375.

STROUBOLIS, T., COPPS, K., AND BABUSKA, I. 2001. The generalized finite element method. *Computer methods in applied mechanics and engineering 190*, 4081–4193.

STROUD, A. H. 1971. *Approximate Calculation of Multiple Integrals*. Prentice-Hall, Englewood Cliffs, New Jersey.

TSUKANOV, I., AND SHAPIRO, V. 2002. The architecture of SAGE – a meshfree system based on RFM. *Engineering with Computers 18*, 4, 295–311.

VARADHAN, G., KRISHNAN, S., SRIRAM, T., AND MANOCHA, D. 2004. Topology preserving surface extraction using adaptive subdivision. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 235–244.

WANG, S., AND WANG, M. 2006. Radial basis functions and level set method for structural topology optimization. *International Journal for Numerical Methods in Engineering 65*, 12 (March), 2060–2090.