TOWARDS THE THEORY AND PRACTICE OF VERIFYING VISUALIZATIONS

by

Tiago Etiene

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Computer Science

The University of Utah

February 2013

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of Tiago Etiene
has been approved by the following supervisory committee members:

Cláudio T. Silva , Chair enter date	Date Approved
Robert M. Kirby , Member	Date Approved
Christopher R. Johnson , Member	Date Approved
Luis Gustavo Nonato , Member	Date Approved
Juliana Freire , Member	Date Approved

ABSTRACT

Abstract text here.

To my beloved parents.

"No amount of experimentation can ever prove me right; a single experiment can prove me wrong."

– Albert Einstein

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Acknowledgement text here.

CHAPTER 1

INTRODUCTION

Today's technology provide unprecedented opportunities to scientists for deriving, expanding, or correcting scientific theories. In the past few decades, we have seen an ever increasing capability of acquire, store, and process data. Simultaneously, the scientific visualization emerged as discipline, and started to play a crucial role in the pipeline of many scientists. In fact, visualization became the means through which scientists explore, evaluate, and present results. Propelled by a vibrant community, visualization techniques have become more widespread, and have successfully been applied in a variety of fields, including medical diagnosis, computational fluid dynamics, weather simulation, among others. The wide range of application, and the uniqueness of each field have further motivated the development of more complete and complex visualization technique, some of which include isosurface extraction, direct volume rendering, flow visualization, to name a few. The amount of work published in each of these areas in the past 20 years is remarkable. Visualization researchers have pushed the boundaries and developed many advanced algorithms: some based on complex mathematical concepts, *e.g.* Morse-Smale complex; or require cutting-edge hardware, *e.g.*, GPU; or have hard practical constraints, *e.g.*, bound on triangle quality; huge input data; etc.

With the increasing complexity and importance of visualization techniques in the scientific pipeline, the reliability of visualization started to attract attention. In the past few years, there was an increase in the number of published articles related to the way humans perceive data, accuracy of visualizations, how to extract and depict uncertainty, how visualization compare to each other, and, more general, how *reliable* are visualization. By "reliable", we include topics of current interest of the visualization community that contributes to increase the expert's confidence in visualizations: uncertainty visualization; uncertainty quantification; evaluation; user perception; and verification.

In this context, the main contribution of this dissertation is to advance the theory and practice of *verification of visualization algorithms and implementation.* We provide

the theoretical background for the verification of visualization algorithms and the results of applying these techniques to several implementations. Before we dive into the details on how to verify visualization algorithms, we go over the motivations behind verifying visualizations. The motivation behind the need for rigorously testing algorithms and implementation comes from several sources: good practice in software engineer; decrease costs [**?**]; safety [**?**, **?**, **?**]; *etc.*. These are the well-known (good) reasons for performing software verification. Here, instead, we argue that at the heart of the methodology for verifying visualization, one will find just good science, which, in our case, means the scientific method. Because visualization has become such an important part in the scientific inquiry, some scientific endeavors, will require softwares to be throughly verified.

1.1 The method of inquiry

New theories are put forward and evaluated through the scientific method: observations; hypothesis formulation; predictions and testing; and analysis of the results. Details on how to perform each of these steps vary according to the phenomena being studied. A particular important step in the process of deriving a valid scientific theory is the process of *falsification*: the process by which the theory risky predictions are tested. From "The Logic of Scientific Discovery" [136]:

> With the help of other statements, previously accepted, certain singular statements – which we may call "predictions" – are deduced from the theory; especially predictions that are easily testable or applicable. From among these statements, those are selected which are not derivable from the current theory, and more especially those which the current theory contradicts. Next we seek a decision as regards these (and other) derived statements by comparing them with the results of practical applications and experiments. If this decision is positive, that is, if the singular conclusions turn out to be acceptable, or verified, then the theory has, for the time being, passed its test: we have found no reason to discard it. But if the decision is negative, or in other words, if the conclusions have been falsified, then their falsification also falsifies the theory from which they were logically deduced.

An example of testing risky prediction is the classic Eddington's experiment of Einstein's theory of relativity [27]. Eddington conducted an experiment to measure the amount of light bending caused by the massive size of the Sun. During the eclipse of 29 May 1919, Eddington photographed the Hyades star cluster and measured the amount light deflected.

The accepted theory at the time, Newton's law of gravity, predicted some shift in the position of the stars. Einstein's theory of gravity predicted twice as much shift as Newton's theory. Because Einstein's prediction contradicted current theory, it can be considered a risky prediction. The eclipse was photographed, and the deviations were measured. In this context, two outcomes are possible: the expected (predicted) did not matched the observed deflection – because no deflection is observed, or Newton's prediction is observed, or some other value is obtained – in which case the theory would be refuted; or, the predicted deflection matched the observations, in which case, nothing can be said about the correctness of the theory, aside from the fact that it has not been proved wrong and has stood to risky tests. The more a theory is tested, the more trustworthy it becomes.

The same idea of falsification can be applied to test the trustworthiness of a computer algorithm and implementation. During the course of scientific inquiry, scientists carefully perform each step in the scientific method in order to mitigate and control errors. For each step of the scientific method, there are multiple ways of account for these errors: by using sophisticated statistical methods; advanced mathematical models; high-precision equipments; redundancy; etc. The reason behind this is that the reliability of the conclusions drawn depends on how each of the steps are performed. In the example of Eddington's experiment, a series of precautions had to be made and several different error sources were taken into account in order to show that Einstein's predictions were correct[1]. Since the scientific methodology is used to increase one's confidence in a particular statement, it can also be applied to the sub steps involved in the formulation of a scientific theory, which, in turn, builds up one's confidence in the theory itself. For instance, how can one reliably make decision based on the results of his/her simulation code? In this case, the ideas behind falsifying a pieces of code have been proved valuable. The need for proving that a determined piece of code is correct is of crucial importance for the scientific pipeline. The lack of such guarantees made the discipline of CFD – which stands for Computational Fluid Dynamics – be once referred to as Colorful Fluid Dynamics [108]. The analog for

[1]They started by sending off two expeditions. The first to Sobral, northern Brazil; and another to the island of Príncipe, northern of São Tomé and Príncipe. Aside equipment related to the telescope, backup lenses were packed along with equipment necessary to account for the rotation of the Earth. The expedition at Príncipe, among other problems, had to deal with clouds and rain and thus they were able to retrieve only two usable photos. The expedition at Sobral, on the other hand, had better weather but had problem due to rise of the temperature between the time that the telescope was assembled and the time of the eclipse. Parts of the telescope expanded and as a result the images were blurred. Another addition problem is related to the very small expected light deflection. Since photograph plates could expand and shrink with the temperature, deflection could be due to other factors other than deflection, such as shrinkage. Other source of errors are involved. According to Coles [27], at the time the results were published, they were met with some skepticism. For more details, see Coles [27].

the visualization literature is the "pretty picture" [50]. Of course, both communities have developed frameworks and practices for dealing with these problem.

1.2 Verification

The meaning word "verification" vary according to the context in which it is inserted. When applied loosely, it refers to good coding practices (such as use of versioning system), software testing (unit/regression tests), and even the process of debugging a code. These practices are obviously valuable for helping build a trustworthy software, but they are often *ad hoc* and have limited scope. In this dissertation, verification of visualization algorithms and implementations is closely related to its use in Computer Science (CS) and Computation Simulation (CS&E). In CS according to IEEE standards, *verification* is the "process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase" [63]. In this context, the program *specification* is the condition imposed at the start phase and the verification process ideally should guarantee that the resulting *implementation* (*i.e.* final computer software) meets the specification exactly. In the past few decades, several techniques have been developed for attaining software verification, which include theorem provers [12], model checking [25], fuzzing [7, 52], and others. This variety of techniques is due to the difficult task of testing a program (either a model or an implementation) which may contain millions of lines of code and an exponentially large state-space where a bug might be hidden. Because of this large number of paths, verification can be considered as a process where one accumulates evidence that a code is correct [139], rather than deriving a proof that the code is actually correct. These techniques have been successfully applied not only for verification of user-level computer code [5], but also hardware [153] and operating system kernel [75].

Verification techniques developed in CS&E community are focused in general on the numerical solution of partial differential equations (PDEs) that models a physical phenomena of interest. In this context, *verification* is defined as the process of determining if a *computational model*, and its corresponding numerical solution, obtained by discretizing the *mathematical model* (with corresponding exact solution) of a physical event and the code implementing the computational model can be used to represent the mathematical model of the event with sufficient accuracy [3]. This definition is closely related to the errors involved during discretization and implementation. They are of great importance for scientists because they can be used to assess which of the models, mathematical or computational, should be refined. A successful approach for code verification, known to

be sensitive to code mistakes [139], is the order of accuracy test — an evaluation of the implementation behavior when submitted to successive grid refinement [139].

The verification tools proposed by the CS and CS&E communities covers only part of the scientific pipeline. Given the importance of the visualization step in the scientific pipeline, the need for reliable visualizations slowly become central to the scientific community. This means that not only numerical software has to be verified, but also visualization algorithms and implementations. Nevertheless, visualization has not fallen under the same rigorous scrutiny as other components of the pipeline like mathematical modeling and numerical simulation. Unlike in traditional computational science and engineering areas, there is no commonly accepted framework for verifying the accuracy, reliability, and robustness of visualization tools. This precludes a precise quantification of the visualization error budget in the larger scientific pipeline. Furthermore, very few investigations have focused on how the error originating from other components of the computational pipeline impacts visualization algorithms.

While the lack of a well-established framework for verifying visualization tools has meant that a variety of analysis techniques have been developed [190, 169], we believe that visualization has achieved sufficient importance to warrant investigation of stricter verification methodologies. Several authors have already asserted this need [50, 51, 72], and in this work we present techniques that are a concrete step towards reducing the gap between best practices in simulation science and visualization.

1.3 Contributions

We advocate that all visualization algorithms and implementation should be verified. Hence, the main contributions to this dissertation is to advance the theory and practice of verifying visualizations. We do so by first extracting important mathematical properties of the algorithms under verification. Then, we verify if the implementation under verification honors that property by stress testing it. As in the case of falsification of scientific theories, a mismatch reveals a problem in the pipeline: the implementation may have a problem; or, perhaps, the algorithm is flawed; or the verification pipeline may have problems. In any case, a problem is revealed. On the other hand, in case the implementation honors the property of interest for all performed tests, then nothing can be said about the correctness of the implementation and algorithm, except that it has stood severe stress test. More specifically, our contribution can be summarized as follows:

1. Chapter 2: verification of geometrical properties of isosurface extraction techniques.

(a) We introduce a framework for verification of visualization tools based on the Method of Manufactured Solutions, and show how to apply it for the verification of geometrical properties of isosurface extraction algorithms and implementations;

(b) We provide the required mathematical analysis for the implementation of the verification procedure. The important properties that should be honored are derived from this mathematical analysis;

(c) We show concrete results, and show how this verification procedure helped us to find and fix problems in both algorithm and implementation. Moreover, we show that it is not trivial to find solutions that are both geometrically accurate and honors additional properties (such as triangle quality);

(d) As a by-product of the verification, we detail the behavior of several freely available implementation;

2. Chapter 3: verification of topological properties of isosurface extraction techniques.

(a) We introduce a framework for verification of topological properties of isosurface extraction algorithms;

(b) We derive a framework based on Digital Topology for the extraction of important invariants (topological properties) that should be honored by topologically correct implmentations;

(c) In addition, we derive a framework based on Stratified Morse Theory for the extraction of important invariants;

(d) We detail the behavior of several freely and commercially available implementations of isosurface extraction; We show that all but one implementation failed our tests;

3. Chapter 4: practical consideration on Marching Cubes 33 topological correctness.

(a) We show that both the Marching Cubes 33 algorithm and implementation have problems that prevents its topological correctness. Moreover, one of the problems is traced back to its original publication;

(b) We propose a new and alternative ways to deal with the issues raised;

   (c) Building on recent efforts on executable papers, we provide new ways to interact with our work so as to improving understanding and reproducibility of the results shown;

4. Chapter 5: verification of volume rendering techniques.

   (a) We introduce a framework for verification of volume rendering algorithms and implementations;

   (b) We provide the error analysis of the standard volume rendering integral that is crucial for the verification procedure;

   (c) We show how we used this information to find and fix problems in widely used volume rendering implementations. Moreover, we provide a first attempt to detect the sensitivity of the verification procedure;

5. Chapter 6: flow visualization and reliable visualizations.

   (a) We provide an overview of the some of the topics involved in reliable visualization. We focus on flow visualization and supporting tools.

The goal of this dissertation is to provide another step towards creating a culture of verification inside the visualization and related communities.

CHAPTER 2

VERIFYING GEOMETRY OF ISOSURFACE EXTRACTION ALGORITHMS

In this chapter, we are concerned with important properties of isosurface extraction algorithms. In particular, in this chapter we will focus on geometrical properties, thus the verification procedure developed in the forthcoming sections will be focused on the correctness of the geometry of the extracted surfaces. This is in contrast to topological properties of isosurfaces, which will be discussed in Chapter 3.

2.1 Related Work

Isosurface extraction is a popular visualization technique, being a tool currently used in science, engineering, and applications. This popularity makes it a natural target for this first application of verification mechanisms in the context of visualization. This same popularity has also driven a large body of work comparing different isosurface extraction algorithms.

Previous researchers have examined topological issues [123, 90], mesh quality [31, 149], accuracy [127, 190], and performance [161]. The influence of different reconstruction schemes and filters in scalar visualization has also been examined [16, 135]. In this paper, we focus on techniques to verify the correctness of algorithms and their corresponding implementations. In particular, we provide mathematical tools that other researchers and developers can use to increase their confidence in the correctness of their own isosurface extraction codes. A traditional way to test implementations in scientific visualization is to perform a visual inspection of the results of the Marschner-Lobb dataset [98]. In the context of isosurface extraction, researchers routinely use tools such as Metro [24] to quantitatively measure the distance between a single pair of surfaces. We argue that the methodology presented here is more effective and more explicit at elucidating a technique's limitations. In particular, our

proposal pays closer attention to the interplay between a theoretical convergence analysis and the experimental result of a *sequence* of approximations.

Globus and Uselton [51] and more recently Kirby and Silva [72] have pointed out the need for verifying both visualization techniques and the corresponding software implementations. In this paper, we provide concrete tools for the specific case of isosurface extraction. Although this is only one particular technique in visualization, we expect the general technique to generalize.

It is important to again stress that verification is a *process*: even when successfully applied to an algorithm and its implementation, one can only concretely claim that the implementation behaves correctly (in the sense of analyzed predicted behavior) for all test cases to which it has been applied. Because the test set, both in terms of model problems and analyzed properties, is open-ended and ever increasing, the verification process must continually be applied to previous and new algorithms as new test sets become available. This does not, however, preclude us from formulating a basic set of metrics against which isosurface extraction methods should be tested, as this is the starting point of the process. This is what we turn to in the next section.

2.2 Verifying Isosurface Extraction Algorithms

In this section, we describe the technique we use for verifying isosurface extraction algorithms, namely the *method of manufactured solutions* (MMS). We illustrate a possible implementation of MMS in Algorithm 1 and Figure 2.2. This technique requires us to write down the expected behavior of particular features of interest of the object (or model problem) being generated. In our case, we are generating triangular approximations of smooth isosurfaces, and the features of interest are geometric surface convergence, convergence of normals, area and curvature.

To use MMS, we first accomplish a mathematical analysis of the expected convergence rate of the features (or characteristics) of interest, known in the numerical literature as the *formal order of accuracy* of the characteristic. This analysis is done for solutions of the problem that can be conveniently described and analyzed (these are the manufactured solutions). Then, the code is executed with progressively refined versions of the data that is used in the generation or sampling of the manufactured solution. Finally, the empirical convergence rate is compared to the one predicted by the analysis. When the convergence rates are comparable, we increase our confidence in the algorithm. If the realizable behavior disagrees with the analysis, either (1) the analysis does not correspond to the correct behavior of the algorithm, (2) the assumptions upon which the analysis was build were violated

Algorithm 1 Overview of the method of manufactured solutions (MMS).

MMS(f, u, h_1)
1 ▷ Let f be a scalar field containing the solution surface S
2 ▷ Let u be a given property (f, normals, area, etc.)
3 ▷ Let h_1 be the initial grid size
4 **for** $i \leftarrow 1$ **to** n
5 **do** $G_{h_i} \leftarrow$ an approximation of f at grid size h_i
6 $S_{h_i} \leftarrow$ an approximation of S computed from G_{h_i}
7 $E_{h_i} \leftarrow ||u(S_{h_i}) - u(S)||_u$
8 $x_i \leftarrow \log h_i$, $y_i \leftarrow \log E_{h_i}$
9 $h_{i+1} \leftarrow h_i/2$
10 $\tilde{q} \leftarrow$ slope of best-fit linear regression of (x_i, y_i)
11 Compare $\tilde{q}$ and q

by the input data and hence the predicted behavior is not valid for the circumstances under investigation, or (3) there are issues with the algorithm or with the implementation of the algorithm (depending on access to source code and algorithmic details, one may not be able to distinguish between these two – algorithmic or implementation – and hence we in this work always consider them together. Given sufficient information, the verification process can help further delineate between these two issues). Notice, however, that all three situations warrant further investigation. In the following sections, we will discuss these issues in more detail. Let us first clarify how we will arrive at theoretical and empirical convergence rates.

For a fixed grid size, we will strive to write the approximation error between the desired isosurface property and its approximation by:

$$E = |u_{approx} - u_{exact}|_u = O(h^p) = \alpha h^p \tag{2.1}$$

where u_{approx}, u_{exact} are the approximated and exact values of a property u, $|\cdot|_u$ is the norm used to compare the approximate and exact property, p is the order of accuracy and α is a constant. Practically speaking, the polynomial expression (2.1) is not very convenient for numerical experimentation, as it is hard to find the value of p from the direct plot of h against E. The standard technique to estimate p is to linearize by working on a log-log scale:

$$\log E = \log(\alpha h^p) = \log \alpha + p \log h. \tag{2.2}$$

Using this linearized version, we estimate p from the slope of the line that best fits the points $(\log h, \log E)$ in a least-squares sense. We use this technique in Section 2.3 when

testing the isosurface codes.

MMS critically depends on an analysis of the order of accuracy of expected solutions. Although this seems quite simple, the order of accuracy under a sensitive norm like $||\cdot||_\infty$ has shown in practice to be very effective in bringing out implementation errors in numerical approximation schemes [142, 3]. In this paper, we show that this analysis is just as effective for isosurface extraction. In addition, we believe the convergence analysis required by MMS is interesting in its own right. As we will discuss in Section 2.4, it helps to shed light on the consequences of implementation choices.

In the context of isosurface methods, manufactured solutions can be built by specifying a "solution surface" to be the exact solution and deriving a scalar field that contains such a solution surface as a level set. The verification methodology then proceeds as following: (1) use the manufactured scalar field as input for the isosurfacing methods, (2) run the methods, and (3) check the output surface against the solution surface (sometimes called the *ansatz solution* within the mathematical verification literature). In many cases, the manufactured scalar field can be derived analytically, making the observed order of accuracy tractable (we give examples in next section).

In what follows, we will derive expected orders of accuracy for several features of surfaces produced by isosurface extraction codes. We keep our assumptions about the actual algorithms to a minimum to maximize the applicability of the arguments given. We essentially only assume that the maximum triangle size can be bounded above at any time, and use Taylor series arguments (under assumptions of smoothness) to derive convergence rates. It is important to point out that order of accuracy analysis of polyhedral surfaces has been studied by many researchers [106, 185, 186, 58]. In fact, the results presented below are in agreement with the ones reported in the literature. However, because we are considering isosurface extraction, some of our arguments benefit by being able to be condensed to simpler statements.

2.2.1 Convergence of Vertex Position

We start our analysis of isosurface extraction by studying the convergence of vertex positions. We analyze this convergence indirectly by relating the values of the scalar field at the vertex points and the distance between the vertices and the correct isosurface. Given a value λ such that the exact isosurface S is defined by $f(x, y, z) = f(\vec{v}) = \lambda$, the *algebraic* distance of $\vec{v}$ to S is defined as $|f(\vec{v}) - \lambda|$ [162]. Notice that algebraic distances only makes sense for implicit surfaces: it requires a scalar field. In addition, we restrict ourselves to *regular* isosurfaces, ones where for every point x in S, $|\nabla f(x)|$ exists and is nonzero. Then,

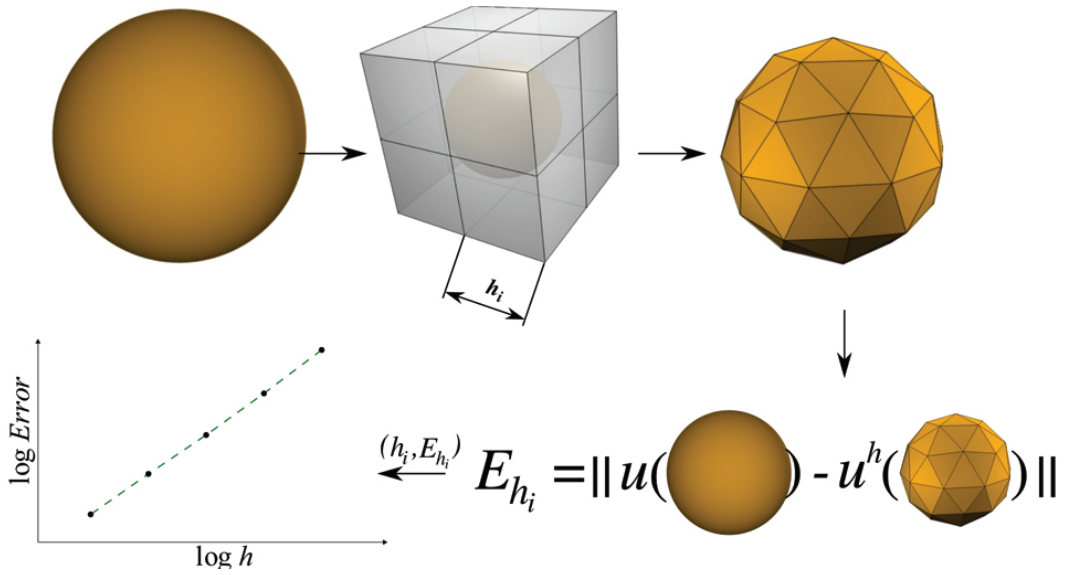


Figure 2.1. Workflow for the method of manufactured solution (Figure 1), clockwise from the top left.

the geometric distance between $\vec{v}$ and S is approximated by $|f(\vec{v}) - \lambda|/|\nabla f(\vec{v})|$ [162]. We illustrate this relation in Figure 2.2. Since, by assumption, $|\nabla f(x)| > k$ for some $k > 0$, and all x in S, convergence in algebraic distance implies convergence in geometric distance. Convergence in algebraic distance, however, is much more tractable mathematically, and this is the item to which we turn our focus.

Many isosurface methods estimate vertex positions through linear interpolation along edges of a grid. Let $f : U \subset \mathbb{R}^3 \to \mathbb{R}$ be the a smooth real function defined in a subset $U = [a_x, b_x] \times [a_y, b_y] \times [a_z, b_z]$, where $[a_i, b_i], i \in x, y, z$ are real intervals. We assume the intervals $[a_i, b_i]$ have the same length and let $a_x = x_0, \ldots, x_n = b_x$, $a_y = y_0, \ldots, y_n = b_y$, and $a_z = z_0, \ldots, z_n = b_z$ be subdivisions for the intervals such that $x_i = x_0 + ih$, $y_i = y_0 + ih$, $z_i = z_0 + ih$, $i = 0, \ldots, n$, where h is the grid size and $c_{ijk} = [x_i, x_{i+1}] \times [y_j, y_{j+1}] \times [z_k, z_{k+1}]$ is a grid cell. Through a Taylor series expansion of f, one can evaluate f at a point $\vec{p} \in c_{ijk}$ as:

$$f(\vec{p}) = f_{ijk} + \nabla f_{ijk} \cdot \vec{\delta} + \frac{1}{2}\vec{\delta}^T H(\vec{\xi})\vec{\delta} \tag{2.3}$$

where $f_{ijk} = f(x_i, y_j, z_k)$, ∇f_{ijk} is the gradient of f in (x_i, y_j, z_k), $H(\vec{\xi})$ is the Hessian of f at a point $\vec{\xi}$ connecting (x_i, y_j, z_k) and $\vec{p}$, and $\vec{\delta} = (u, v, w)^T$ is such that $\vec{p} = (x_i + uh, y_j + vh, z_k + wh)^T$.

Let the linear approximation of f in $\vec{p}$ be defined by

$$\tilde{f}(\vec{p}) = f_{ijk} + \nabla f_{ijk} \cdot \vec{\delta} \tag{2.4}$$

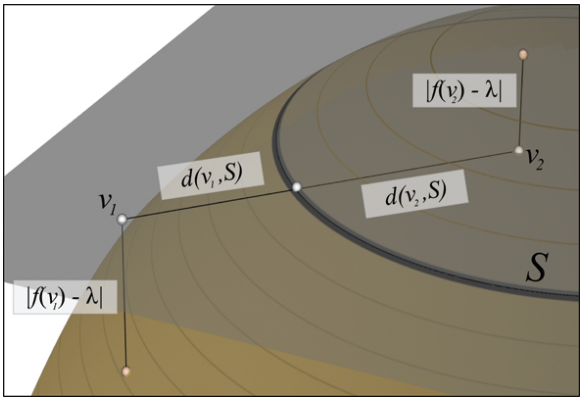


Figure 2.2. The distance between a point $\vec{v}$ and the isosurface S with isovalue λ can be approximated by the algebraic distance divided by the gradient magnitude of the scalar field at $\vec{v}$, $|f(\vec{v}) - \lambda|/|\nabla f(\vec{v})|$. In the figure, the thick circle represents the isosurface S and the fainter isolines illustrate changes in gradient magnitude: in regions of small gradient magnitude, the algebraic distance is small but geometric distance is large, and vice-versa for large gradient magnitude.

and consider a point $\vec{x_\lambda}$ such that $\tilde{f}(\vec{x_\lambda}) = \lambda$, that is, $\vec{x_\lambda}$ is a point on the isosurface λ of $\tilde{f}$.

The algebraic distance between the exact isosurface $f(x, y, z) = \lambda$ and the linearly approximated isosurface can be measured by $|f(\vec{x_\lambda}) - \lambda|$. From Equations 2.3 and 2.4 one can see that

$$\begin{gathered} |f(\vec{x_\lambda}) - \lambda| = |f_{ijk} + \nabla f_{ijk} \cdot \vec{\delta} + \frac{1}{2}\vec{\delta}^T H(\vec{\xi})\vec{\delta} - \lambda| = \\ |\tilde{f}(\vec{x_\lambda}) + O(h^2) - \lambda| = O(h^2) \end{gathered} \tag{2.5}$$

thus, the linearly approximated isosurface is of second-order accuracy.

2.2.2 Convergence of Normals

Assume, generally, that the scalar field $f(x, y, z) = \lambda$ can be locally written as a graph of a function in two-variables $g(x(u, v), y(u, v)) = \lambda - f(x(u, v), y(u, v), z_k)$, as illustrated in Figure 2.3, where $x(u, v) = x_i + uh$ and $y(u, v) = y_j + vh$. This is acceptable because we have already assumed the isosurface to be regular. Still without losing generality we write $g(x(0, 0), y(0, 0)) = 0$, that is, the isosurface contains the point (x_i, y_j, z_k). Let $\vec{\Phi}(u, v) = (x(u, v), y(u, v), g(x(u, v), y(u, v)))$ be a parametrization for the isosurface $f(x, y, z) = \lambda$ in c_{ijk} and

$$\frac{\partial\vec{\Phi}}{\partial u} \times \frac{\partial\vec{\Phi}}{\partial v} = h^2 \left(-\frac{\partial g}{\partial x}, -\frac{\partial g}{\partial y}, 1 \right)^T = h^2\vec{n_0} \tag{2.6}$$

be the normal vector in $\vec{\Phi}(0, 0) = (x_i, y_j, g(x_i, y_j))$ (the partial derivatives of g are evaluated at $(x(0, 0), y(0, 0)) = (x_i, y_j)$).

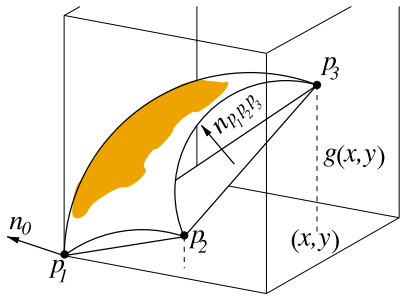


Figure 2.3. Isosurface local parametrization and approximation.

Consider now the triangle defined by the points $\vec{p_1}, \vec{p_2}$, and $\vec{p_3}$ approximating the isosurface $f(x, y, z) = \lambda$ in the grid cell c_{ijk} (see Figure 2.3). Let $\vec{p_1}$ be the grid point (x_i, y_j, z_k), so $\vec{p_1} = \vec{\Phi}(0, 0)$, $\vec{p_2} = \vec{\Phi}(u_2, v_2)$, and $\vec{p_3} = \vec{\Phi}(u_3, v_3)$. Using the cross product in $\mathbb{R}^3$, the normal of the triangle $p_1p_2p_3$ can be computed by:

$$\begin{gathered} n_{p_1\vec{p}_2p_3}=(\vec{p_2} - \vec{p_1}) \times (\vec{p_3} - \vec{p_1})= \\ \begin{pmatrix} h(v_2 g(x(u_3, v_3), y(u_3, v_3)) - v_3 g(x(u_2, v_2), y(u_2, v_2))) \\ h(u_3 g(x(u_2, v_2), y(u_2, v_2)) - u_2 g(x(u_3, v_3), y(u_3, v_3))) \\ h^2(u_2 v_3 - u_3 v_2)) \end{pmatrix}. \end{gathered} \tag{2.7}$$

Expanding $g(x(u_i, v_i), y(u_i, v_i))$, $i \in \{2, 3\}$ in a Taylor series, some terms cancel and the normal $n_{p_1\vec{p}_2p_3}$ becomes:

$$n_{p_1\vec{p}_2p_3} = rh^2 \left(-\frac{\partial g}{\partial x} + O(h),\ -\frac{\partial g}{\partial y} + O(h), 1 \right)^T \tag{2.8}$$

where $r = u_2v_3 - u_3v_2$. Comparing the exact normal vector $\vec{n_0}$ in Equation 2.6 with $n_{p_1\vec{p}_2p_3}$ above, we recover first-order of accuracy for normals. In addition, notice that the usual scheme of estimating vertex normals by the arithmetic mean of triangle normals does not decrease the order of accuracy; that is, vertex normals (computed by arithmetic mean) are at least first-order accurate.

2.2.3 Convergence of Area

Currently, much less is known about convergence in area, compared to convergence of vertices or normals. To illustrate the difficulty involved in approximating lengths and areas, consider the sequence of approximations to a straight line shown in Figure 2.4. Even though the function sequence converges uniformly to the line, the length of the approximation stays constant.

To the best of our knowledge, the only relevant results establish convergence in area given convergence in vertex positions *and* convergence in normals, such as in Hildebrandt *et al.* [58]. However, the authors only establish asymptotic convergence, with no order of

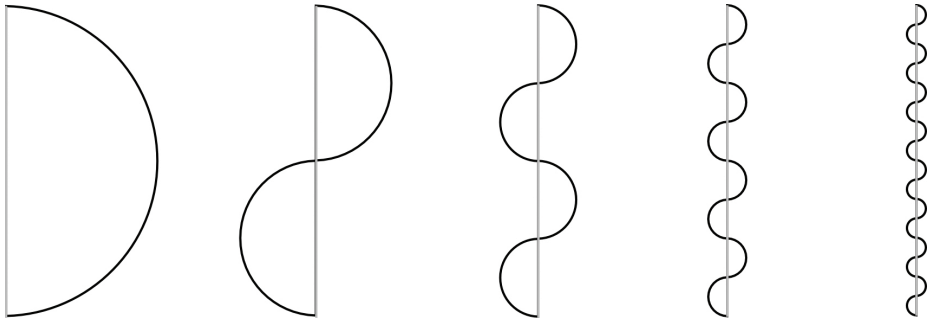


Figure 2.4. Uniform convergence does not imply convergence in area. The sequence of curves converges uniformly to a straight line, but the length of the curves does not change.

accuracy associated with it. The argument is more mathematically involved than space allows here, so we refer the reader to that paper. Currently, this means that the only information the observed order of accuracy provides to us is that if we expect convergence in normals, we should also expect convergence in area, and vice-versa.

2.2.4 Convergence of Curvature

The following formula gives an estimate of the curvature at a vertex p:

$$K(p) = \frac{2\pi - \sum \theta_{i\,i+1}}{\frac{1}{3}A_{i\,i+1}} \tag{2.9}$$

where $\theta_{i\,i+1}$ and $A_{i\,i+1}$ are the angle $\angle p_i p p_{i+1}$ and area of the triangle $p_i p p_{i+1}$ respectively (summation is over all triangles comprising the star of p) [106]. Meek and Walton [106] showed that the curvature computed via Equation 2.9 does not converge in general; that is, if the vertices of the star of p are arbitrarily distributed around p, one cannot expect curvature convergence. In fact, they described a more general result stating that $O(h)$ accuracy can only be obtained if the normals are known to have accuracy $O(h^2)$. Subsequently, Xu [185] presented a very particular distribution of vertices around p under which the curvature estimated by Equation 2.9 has accuracy $O(h^2)$.

Curvature discretization schemes other than the one given in Equation 2.9 such as the quadratic-fit and spherical-image method (see Meek and Walton [106] for details) also demand particular vertex distributions to ensure convergence. In our context of keeping the analysis applicable for many isosurfacing algorithms, this means we cannot use the lack of observed curvature convergence as an indication of problematic behavior. Based on the results mentioned above, one should actually expect curvature not to converge for most isosurface extraction algorithms. More generally, this indicates a weakness of MMS, namely that some features of interest (such as curvature) will not have sufficient theoretical

order of accuracy to be used in numerical measurements. Notice, in addition, that if we had not written down the theoretical model for curvature convergence, we might have expected some sort of curvature approximation. Even a negative result such as the one presented in this section can increase the confidence in the results generated by an implementation.

2.3 Experimental Results

In this section we present the results of applying the afore-described methodology. We use the framework to verify six different isosurface extraction codes, namely: VTK Marching Cubes [96], SnapMC [137], Macet [31], Dual Contouring [69], Afront [149], and DelIso [30]. All these implementations are open source and/or publicly available[1]. Before presenting the actual results of subjecting these implementations to the verification process, we briefly review their salient features.

VTK Marching Cubes: Marching Cubes [96] (MC) is arguably the most popular isosurface extraction algorithm. It reduces the problem of generating an isosurface triangulation to a finite set of cases by considering the *signs* of how the isosurface intersects each cell of a regular background grid. As there are only 256 different types of intersections between the isosurface and a regular Cartesian 3D cell, a template of triangles is set to each case, making the implementation quite simple through a look-up table. The vertices of the triangles lie on the edges of the cubic cells, and they are computed by linearly interpolating the implicit function values stored at the corners of the grid cell.

SnapMC: SnapMC [137] is a recently proposed algorithm that extends the original Marching Cubes look-up table to cases where the isosurface goes exactly through the corners of the background grid. The new look-up table is automatically built by an adaptation of the convex hull scheme proposed by Bhaniramka *et al.* [6]. Even though the traditional Marching Cubes algorithm can easily handle these cases by some kind of symbolic perturbation, SnapMC *perturbs the scalar field* to avoid edge intersections close to grid corners. In particular, it changes the values on the grid so that the surface is "snapped" to the grid corners.

Macet: Macet [31] is another variant of Marching Cubes that tries to improve the shape of the triangles in a mesh. Unlike SnapMC, it *perturbs the active edges* of Marching Cubes cases by moving the vertices before the triangulation step. The motivation behind Macet is that poorly-shaped triangles tend to be generated when the intersection between the isosurface and a grid cell is approximately parallel to an edge of the grid cell. There-

[1]Links at http://www.sci.utah.edu/~etiene/

fore, some corners of the background grid are displaced so as to avoid the parallel-like intersections.

Dual Contouring: Dual Contouring [69] is a feature-preserving isosurfacing method to extract crack-free surfaces from both uniform and adaptive octree grids. This technique can be seen as a combination of Extended Marching Cubes [77] and SurfaceNets [49] as it makes use of Hermite data and quadratic error function minimization to position the vertices of the surface mesh (as Extended Marching Cubes) and the dual topology to connect such vertices (as SurfaceNets). Dual Contouring tends to generate better quality triangles than Marching Cubes while still being very effective in representing sharp features, rendering this implicit polygonalization method a good alternative to the popular Marching Cubes.

Afront: Afront [149] is an advancing-front method for surface extraction. Although we focus on applying Afront to isosurface extraction, it can also be used for remeshing and triangulating point-set surfaces. The outstanding feature of Afront is that it generates triangles adapted to the local details of a surface, namely its maximum absolute curvature. In this sense, Afront is fundamentally different from the other algorithms we analyze. In lieu of grid refinement, we will use its ρ parameter to control triangulation size. Because the manufactured solution we use is a sphere, reducing ρ by half is roughly equivalent to reducing the maximum triangle size by half. A full analysis of Afront (and, in particular, the influence of the other main parameter η) warrants further investigation, but is beyond the scope of this paper.

DelIso: DelIso [30] is a Delaunay-based approach for isosurfacing. It computes the restricted Delaunay triangulation from a 3D Voronoi Diagram. We run our tests on a customized version of DelIso 16 bit, and our examples use the default set of parameter.

In what follows, we present the results of applying the verification process to these algorithms. We will describe the manufactured solutions we use and their observed convergence rate on the isosurface extraction algorithm.

2.3.1 Observed order of accuracy

We start by investigating the behavior of the algorithms under the manufactured solution given by the scalar field $f(x, y, z) = x^2 + y^2 + z^2 - 1$ and isosurface $f(x, y, z) = 0$ in the domain $D = [-4, 4]^3$. Let $\tilde{S}_k$ be a simplicial complex that approximates S for a given discretization parameter k (cell size h for marching cubes-based methods, accuracy ρ for Afront and maximum edge size ι for DelIso).

The order of accuracy for VTK Marching Cubes, SnapMC, Macet and Dual Contouring depends on the cell size h. We run our tests with grid refinement $h_{i+1} = h_i/2$ and initial

condition h_1. For Afront, the order of accuracy depends on parameter ρ thus the refinement is given by $\rho_{i+1} = \rho_i/2$ with initial condition ρ_1. Our customized version of DelIso has an additional parameter ι that controls the largest edge on the output mesh. In this case, the refinement formula is $\iota_{i+1} = \iota_i/2$. In the particular case of SnapMC, we set the snap parameter γ to its maximum value ($\gamma = 1/2$). Even though the manufactured solution we selected is about as simple as can be imagined, comparing the formal order of accuracy with the observed one was enough to suggest bugs in two implementations. The observed order of accuracy of the examined properties is presented on Table 2.1.

2.3.1.1 Algebraic distance

Section 2.2.1 shows that one expects second-order convergence for function value on vertices if linear interpolation is used. We define the following approximation error on L_∞ norm:

$$E_k = \max_{j=1\cdots n} |\lambda - f(v_j)| \tag{2.10}$$

where λ is the isovalue of interest, v_j is a vertex of $\tilde{S}$ and n the number of vertices. Figure 2.5(a) shows the vertex observed order of accuracy. VTK Marching Cubes, SnapMC have nearly quadratic convergence rates as shown in Figure 2.5(a). Afront shows a zero-order of accuracy though it presents very low error (in fact, the lowest in Figure 2.5(a)). This is due to the Catmull-Rom spline that is being used for surface approximation on the voxelized grid. Since it has cubic-order of accuracy, even for large values of ρ it can approximate with high precision the manufactured solution f. Next section shows that this is due to a poor

	Vertex $O(h^2)$	Normal $O(h)$	Area –	Curvature $O(1)$
VTK MC	1.94	0.93	2.00	−3.35
SnapMC	1.93	0.82	2.14	−0.29
Afront*	−0.06	0.80	1.93	−0.27
Macet[1,*]	0.98	−0.12	0.29	−2.41
Macet[2,*]	0.03	0.75	2.02	−0.61
DC[1]	1.02	−0.11	0.69	−2.08
DC[2]	1.96	0.96	1.89	−0.15
DellIso	1.49	1.07	2.04	0.07

Table 2.1. Comparison between formal order of accuracy and observed order of accuracy using $f(x, y, z) = x^2 + y^2 + z^2 - 1$ as a manufactured solution and for different algorithms. [1] indicates the original source code and [2] our fixed version. * indicates that a high-order spline was used instead of a linear interpolation (Section 2.2).

choice for a manufactured solution. DelIso implementation has non-zero order of accuracy due to an outlier. Large values of ι causes bad approximations of the manufactured solution.

The Macet and Dual Contouring curves suggest that the algorithms converge to a fixed value. In fact, there was indeed a problem in the implementation that was affecting the convergence of Macet and Dual Contouring (specifically, we found a hard-coded limit in the number of steps in a root-finding procedure that was being triggered by the high resolution of the volume). Once fixed, we obtain the results shown in Figure 2.6(a). Macet and Afront now have similar behavior in the observed order of accuracy of vertex position (Figure 2.6(a)). This is because both methods use high-order interpolation with splines, not linear interpolation as assumed before (see Section 2.4).

2.3.1.2 Normals

Section 2.2.2 shows that one expects first-order of accuracy for normal computations. We define the following approximation error using L_∞ norm:

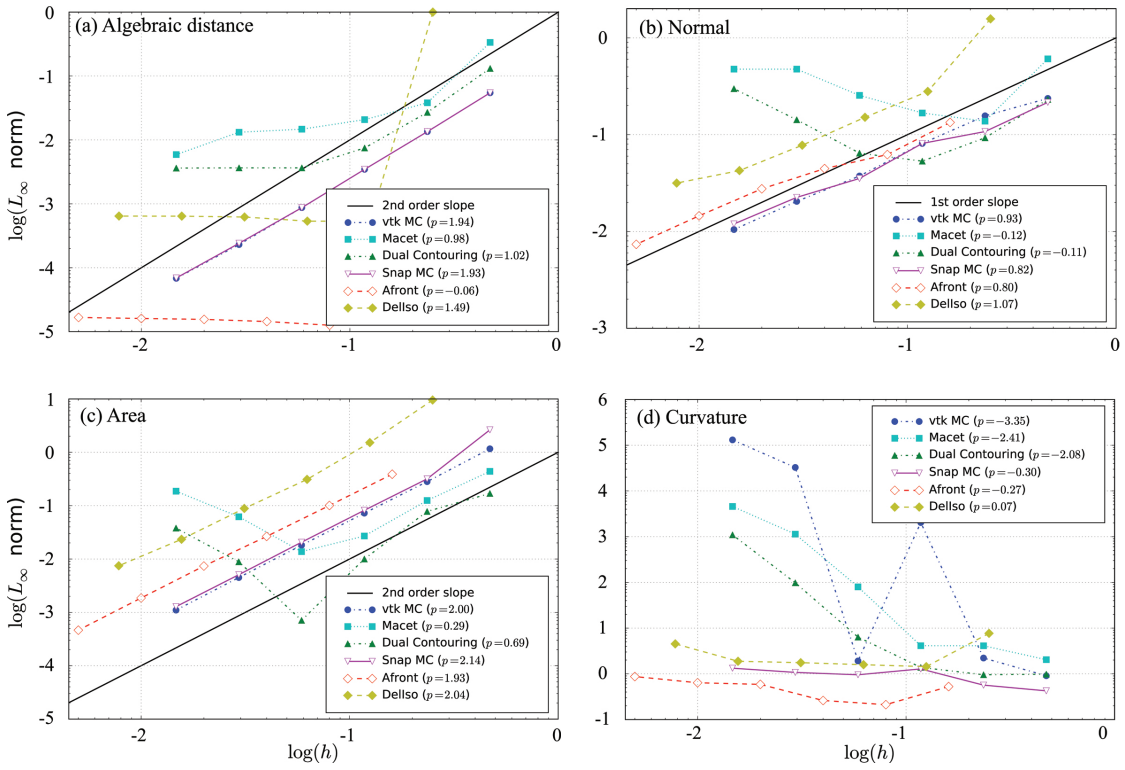


Figure 2.5. Observed order of accuracy. The implementations of Macet and Dual Contouring have a bug that causes the deviation on errors. The black continuous line represents the expected behavior. p is the slope of the linear regression for each curve.

$$E_k = \max_{j=1\cdots n} |\theta_{\sigma_j}| \tag{2.11}$$

where θ_{σ_j} is the angle between the normal of the triangle σ_j and the normal of the point in S closest to the centroid of σ_j. As shown in Figure 2.5(b), VTK Marching Cubes, Afront, SnapMC and DelIso have good observed order of accuracy above 0.8. However, only VTK Marching Cubes and DelIso present close proximity to linear. Macet and Dual Contouring once again do not present a consistent order. Figure 2.6(b) shows the results after fixing both codes.

2.3.1.3 Area

Although there is no formal order of accuracy for area, one expects *some* convergence for it (Section 2.2.3). We define the following approximation error:

$$E_k = |A(S) - A(\tilde{S}_k)| \tag{2.12}$$

where A is the area function of a continuous or piecewise-linear surface. The results are shown in Figure 2.5(c). VTK Marching Cubes, Afront and DelIso present second-order of accuracy as shown in Figure 2.5(c). SnapMC accuracy is slightly better than quadratic due to poor approximation for large h. The error dropped faster than quadratic when the grid was refined for the first time. Macet and Dual Contouring exhibit once again unexpected behavior. Unlike the previous time, the curves now seem to diverge when h is too small. Once the bug is fixed the convergence curves changes, and they become quadratic (Figure 2.6(c)).

2.3.1.4 Curvature

Section 2.2.4 shows that one expects zero-th order of accuracy for curvature computation. We define the approximation error using L_∞ norm:

$$E_k = \max_{j=1\cdots n} |K(v_j) - \tilde{K}(v_j)| \tag{2.13}$$

where $K(v)$ is the Gaussian curvature at $v \in S$ and $\tilde{K}(v)$ is the Gaussian curvature at $v \in \tilde{S}$. In this particular case where S is a sphere, $K(v) = 1$ for every $v \in S$. The results are shown in Figure 2.5(d). DelIso, Afront and SnapMC are close to zeroth-order accuracy. The curvature order of accuracy for VTK Marching Cubes, on the other hand, diverges significantly. This unexpected behavior might deserve further investigation which we leave for future work. Although the curves shown in Figure 2.5(d) for Macet and Dual Contouring diverge, they change after fixing the code (Figure 2.6(d)).

2.3.2 Detected Bugs

We were able to find and fix bugs in two of the implementations under verification, namely, Macet and Dual Contouring, using as manufactured solution a sphere centered at origin with radius 1. The new result curves are shown in Figure 2.6. The observed order of accuracy for Dual Contouring is quite satisfactory for all manufactured solution. In particular, the normal order of accuracy has the best rate among the methods. Macet improved for its results for area. On the other hand, it still has some issues related to normals, which perhaps indicates a need for more tests and verification. The new order of accuracy for algebraic distance (Figure 2.6(a)) does not tell us much about the correctness of the code because of the zero-th order of accuracy (same for Afront).

The zero-th order of accuracy might happen if the formal order of accuracy is zero-th order, in which case the observed order matches the formal order. It might also happen due to a poor choice for manufactured solution. If it is not complex enough, the implementation being tested may approximate exactly the solution and therefore there is no error within the approximation although another error source (truncation error, for instance) may show

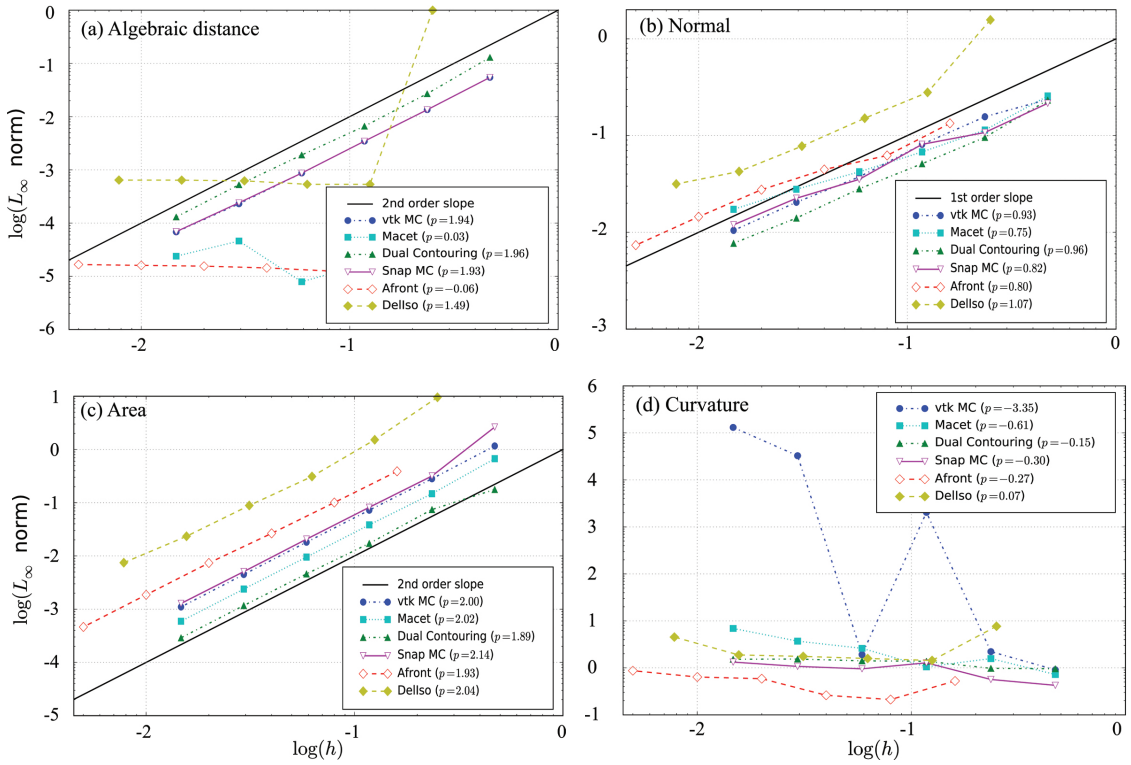


Figure 2.6. Observed order of accuracy after fixing Macet and Dual Contouring code (other curves remain the same). The black continuous line represents the expected behavior. p is the slope of the linear regression for each curve.

up. The next section presents a detailed discussion concerning MMS.

Although we managed to fix the Macet convergence problem, we were not able to do so in a way that preserves triangle quality (Figure 2.7). Two were the problems we found in the source code, and we proposed two solutions for one of them. Table 2.2 shows that we could not find any combination that both fixed the convergence problem and preserved the triangle quality simultaneously. This sort of behavior raises the question if there is a theoretical problem that prevents both from being satisfied simultaneously, or it is just a matter of finding a better algorithmic fix. In both cases, further study and subsequent tests must be accomplished.

2.4 Discussion

As we have shown, MMS is an effective means of diagnosing problems within the algorithms and implementations of isosurface extraction algorithms. In this work we have considered the two – algorithm and implementation – as one unit as one cannot always distinguish between the two if only limited information (source code and algorithmic details) is available. In this section, we present a more thorough discussion of the use of MMS, particularly for isosurface extraction.

On the implementation and use of MMS. One of the primary advantages of verifying simulation codes using MMS is that it is a non-intrusive method. MMS treats

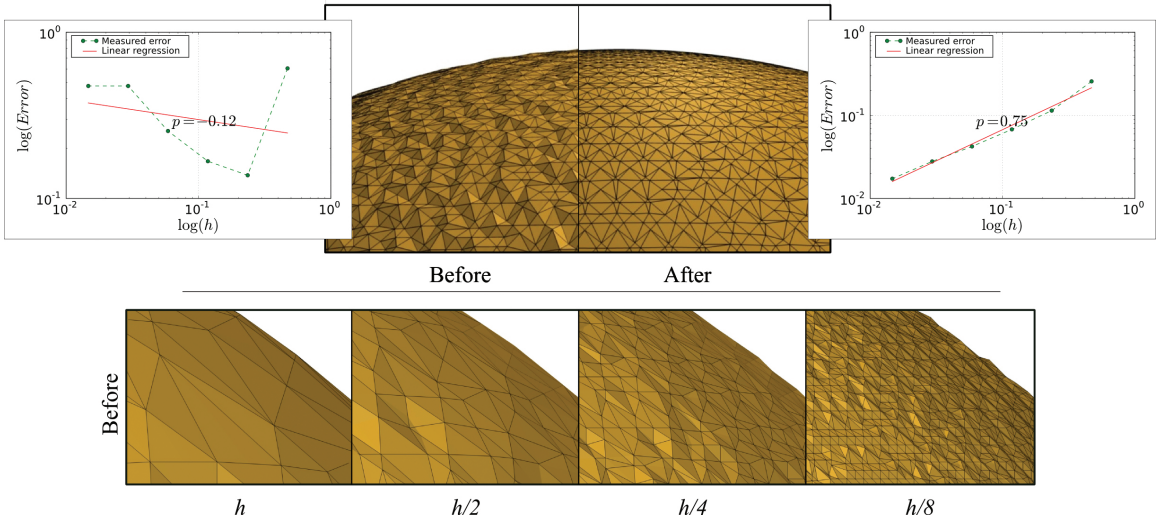


Figure 2.7. Through the verification methodology presented on this paper we were able to uncover a convergence problem within a publicly available marching-based isosurfacing code (top left) and fix it (top right). The problem causes the mesh normals to *disagree* with the known gradient field when refining the voxel size h (bottom row). The two graphs show the convergence of the normals before and after fixing the code.

Bug #1	Bug #2	Quality	Observed accuracy
No Fix	No Fix	Good	Bad
Fix 1	No Fix	Good	Bad
Fix 1	Fixed	Bad	Good
Fix 2	No Fix	Good	Bad
Fix 2	Fixed	Bad	Good

Table 2.2. Table of results for Macet. Triangle quality versus convergence. We were not able to find a solution that provides both triangle quality and convergence.

the code being verified as a blackbox, and so can be easily integrated into an existing test suite with little to no impact. However, MMS does not "see" the implementation, and so provides little direct information about where a particular bug might be when there is a discrepancy between the formal and observed orders of accuracy. In our experience, there are three main places where mistakes can happen: (1) in the design and construction of the manufactured solution, (2) in the coding of the algorithm being tested, and (3) in the evaluation and interpretation of the results. Mistakes on the evaluation of results have two flavors: misinterpretation or poor formal order of accuracy. The first heavily depends on testers' and experts' experience and ability to judge what a good result is. For example, should the normal observed order of accuracy for Afront and Macet on Figure 2.5(b) be considered linear ($p = 0.80$ and $p = 0.75$ respectively)? The latter depends on a rigorous formal order of accuracy analysis of the algorithm considering all sorts of errors; even round-off errors may be significant. In fact, we spent more time on writing out rigorously the analysis of the formal order of accuracy and on searching for possible sources of error than on the tests themselves. This again highlights the fact that verification using MMS is a process: it is typical to go back to the white board and refine formal analyses before arriving at conclusive answers. Although the formal order of accuracy analysis might be a painful process, the literature has many results that can be promptly used. As a consequence, if one wishes to writes his own MC technique, for instance, his only concern is to write a test which exploits the results available within the literature.

On the complexity of the manufactured solution. The complexity of the manufactured solution can have a large influence on the effectiveness of verification. Suppose one chooses the manufactured solution to be $f(x, y, z) = x + y + k$, k constant, instead of a sphere. Since MC-based techniques use linear interpolation, one expects the approximation to be exact regardless of any discretization parameter h, *i.e.*, $p = 0$ (notice that the evaluated error might be non-zero, implying there is some other error source that does not depend on h). Since such a function f is extremely simple, it might not trigger bugs

that would otherwise reduce the observed order of accuracy. In our experiments, the (problematic) implementation of Dual Contouring achieved the formal order of accuracy for this particularly simple function ($p = 0$).

Another example on the influence of manufactured solution arose with in our examination of Afront. Because Afront uses Catmull-Rom splines, some simple isosurfaces will converge to within numerical error for very rough volumes, and the numerically observed order of accuracy will be much lower than expected. With an implicit function whose isosurfaces are spheres, we observed zero-th order of accuracy for Afront for algebraic distance. With a modified implicit function that included transcendental functions, MMS reveals that Afront does not have the expected convergence rate on the full interval, as shown in Figure 2.8. Notice that Macet has similar behavior. Additional tests are needed to determine the source of this behavior within both codes.

On the order of accuracy. In this paper, we have chosen to make our formal analysis as generic as possible to accommodate as many implementations under verification as possible. Although we are able to evaluate many codes using the same manufactured solution, when using MMS for a particular code, it is best to exploit as much detail about the algorithm as necessary. If the goal is to design a manufactured solution for verifying Marching Cubes-based techniques the manufactured solution should exercise all possible cases. Additionally, particular aspects of the manufactured solutions can be incorporated into the formal analysis. For example, the analysis for Afront becomes much more compli-

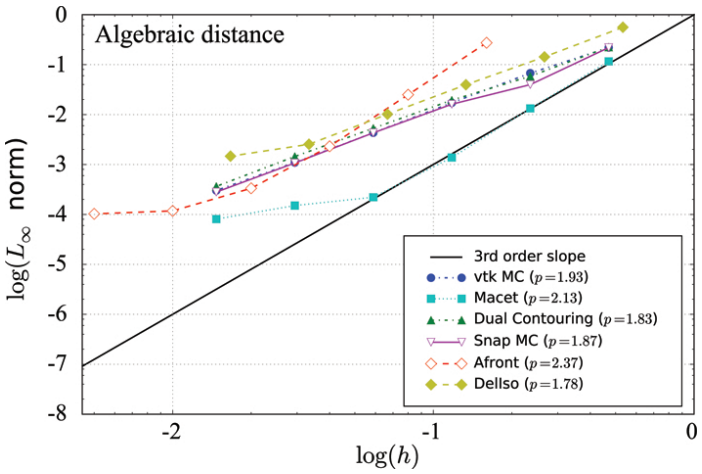


Figure 2.8. Order of accuracy for a transcendental function $f(x, y, z) = x^2 + y^2 + z^2 + \cos(Ax)^2 + \cos(Ay)^2 + \cos(Az)^2$, A is a constant. The observed orders of accuracy for all implementations are relative to the voxel size h. We expect third-order accuracy for Afront and Macet due to their use of high-order spline approximations. Both have the expected convergence rate for all but the last two values.

cated if curvatures are not constant over the surface (in that case, its additional parameter η comes into play [149], and accurately bounding the triangle size is not practical).

The errors in Section 2.3.1 were measured at different locations on the mesh. Vertex convergence and Gaussian curvature were measured on triangle vertices, while normals were measured on the triangle centroid. More importantly, measurements performed at different locations may have different orders of accuracy. For example, Macet has cubic formal order of accuracy on vertices due to the spline approximation but quadratic formal order of accuracy on centroids.

In Section 2.2, we define the error using a pessimistic L_∞ norm. This makes MMS a very sensitive technique. In fact, it can detect subtle off-by-one mistakes in grid sizes and interactions between node-centric and cell-centric reconstructions, even for simple manufactured solutions. In these cases, it is important not to infer incorrect conclusions.

The numerical estimates for MMS should be performed on as wide a range of parameter values as possible. In our tests, we used $h \in (0.001, 1.0)$ and observed that both faulty implementations performed appropriately for large values of h. Just as the implementations might only enter the asymptotic regime and achieve the formal convergences for small values of h, it might be that (as we have experienced) bugs only manifest themselves on sufficiently small values of h.

On the limitations of the test. MMS does not cover every aspect of verification for isosurface extraction. For example, an important aspect we do not know how to test with MMS is the topological correctness of an extracted mesh. This is challenging because there does not seem to be a good measure of convergence for topological properties such as the Euler characteristic or Betti numbers. A proper study of these issues is a natural avenue for future work.

2.5 Conclusions and Future Work

Because of its simplicity and effectiveness, we believe MMS could become a standard tool in the verification of scientific visualization software in the same that it has been adopted by the scientific simulation community as a trustworthy tool for assess code correctness. Using a simple manufactured solution, we were able to reveal bugs that prevented the convergence of some mesh properties of two publicly available isosurfacing codes. In particular, the by-products of the verification process, namely a continuous refinement of mathematical analysis of the algorithm's behavior and a numerical comparison of the results of the implementation against a known solution are valuable in their own right, and should be published together with new algorithms.

We are investigating the applicability of MMS to other visualization techniques such as streamline generation and volume rendering. In particular, MMS should clarify assumptions and errors intrinsic in these visualizations, a topic that has received recent attention[67]. More importantly, we hope the examples presented here will encourage the adoption of MMS by the visualization community at large, increasing the impact of its contributions to a wider audience.

CHAPTER 3

VERIFYING TOPOLOGY OF ISOSURFACE EXTRACTION ALGORITHMS

Visualization is an important aspect of current large-scale data analysis. As the users of scientific software are not typically visualization experts, they might not be aware of limitations and properties of the underlying algorithms and visualization techniques. As visualization researchers and practitioners, it is our responsibility to ensure that these limitations and properties are clearly stated and studied. Moreover, we should provide mechanisms which attest to the correctness of visualization systems. Unfortunately, the accuracy, reliability, and robustness of visualization algorithms and their implementations have not in general fallen under such scrutiny as have other components of the scientific computing pipeline.

The main goal of verifiable visualization is to increase confidence in visualization tools [72]. Verifiable visualization tries to develop systematic mechanisms for identifying and correcting errors in both algorithms and implementations of visualization techniques. As an example, consider a recent scheme to check geometrical properties of isosurface extraction [42]. By writing down easily checkable convergence properties that the programs should exhibit, the authors identified bugs in isosurfacing codes that had gone undetected.

We strive for verification tools which are both *simple* and *effective*. Simple verification methods are less likely to have bugs themselves, and effective methods make it difficult for bugs to hide. Alas, the mathematical properties of an algorithm and its implementation are both constructs of fallible human beings, and so perfection is an unattainable goal; there will always be the next bug. Verification is, fundamentally, a *process*, and when it finds problems with an algorithm or its implementation, we can only claim that the new implementation behaves more correctly than the old one. Nevertheless, the verification process clarifies *how* the implementations fail or succeed.

In this work, we investigate isosurfacing algorithms and implementations and focus on their *topological properties*. For brevity, we will use the general phrase "isosurfacing" when we refer to both isosurfacing algorithms and their implementations. As a simple example, the topology of the output of isosurface codes should match that of the level set of the scalar field (as discussed in Section 3.2). Broadly speaking, we use the method of manufactured solutions (MMS) to check these properties. By manufacturing a model whose known behavior should be reproduced by the techniques under analysis, MMS can check whether they meet expectations.

Etiene et al. have recently used this method to verify geometrical properties of isosurfacing codes [42], and topological verification naturally follows. An important contribution of this work is the selection of significant topological characteristics that can be verified by software methods. We use results from two fields in computational topology, namely digital topology and stratified Morse theory.

In summary, the main contributions of this work can be stated as follows:

1. In the spirit of verifiable visualization, we introduce a methodology for checking topological properties of publicly and commercially available isosurfacing software.

2. We show how to adapt techniques from digital topology to yield simple and effective verification tools for isosurfaces without boundaries.

3. We introduce a simple technique to compute the Euler characteristic of a level set of a trilinearly interpolated scalar field. The technique relies on stratified Morse theory and allows us to verify topological properties of isosurfaces with boundaries.

4. We propose a mechanism to manufacture isosurfaces with non-trivial topological properties, showing that this simple mechanism effectively stresses isosurfacing programs. As input, we also assume a piecewise trilinear scalar field defined on a regular grid.

The verification process produces a comprehensive record of the desired properties of the implementations, along with an objective assessment of whether these properties are satisfied. This record improves the applicability of the technique and increases the value of visualization. We present a set of results obtained using our method, and we report errors in two publicly-available isosurface extraction codes.

3.1 Related Work

The literature that evaluates isosurface extraction techniques is enormous, with works ranging from mesh quality [31, 149, 137], to performance [161] and accuracy analysis [127,

190]. In this section, we focus on methods that deal with topological issues that naturally appear in isosurfacing.

Topology-aware Isosurfacing. Arguably the most popular isosurface extraction technique, Marching Cubes [96] (MC) processes one grid cell at a time and uses the *signs* of each grid node (whether the scalar field at the node is above or below the isovalue) to fit a triangular mesh that approximates the isosurface within the cell. As no information besides the signs is taken into account, Marching Cubes cannot guarantee any topological equivalence between the triangulated mesh and the original isosurface. In fact, the original Marching Cubes algorithm would produce surfaces with "cracks," caused by alternating vertex signs along a face boundary, which lead to contradicting triangulations in neighboring cells [122]. Disambiguation mechanisms can ensure crack-free surfaces, and many schemes have been proposed, such as the one by Montani et al. [113], domain tetrahedralization [16], preferred polarity [8], gradient-based method [175], and feature-based schemes [60]. The survey of Newman and Yi has a comprehensive account [120]. Although disambiguation prevents cracks in the output, it does not guarantee topological equivalence.

Topological equivalence between the resulting triangle mesh and the isosurface can only be achieved with additional information about the underlying scalar field. Since function values on grid nodes are typically the only information provided, a reconstruction kernel is assumed, of which trilinear reconstruction on regular hexahedral grids is most popular [121]. Nielson and Hamann, for example, use saddle points of the bilinear interpolant on grid cell faces [122]. Their method cannot always reproduce the topology of trilinear interpolation because there remain ambiguities internal to a grid cell: pairs of non-homeomorphic isosurfaces could be homeomorphic when restricted to the grid cell faces. This problem has been recognized by Natarajan [118] and Chernyaev [21], leading to new classification and triangulation schemes. This line of work has inspired many other "topology-aware" triangulation methods, such as Cignoni et al.'s reconstruction technique [22]. Subsequent work by Lopes and Brodlie [94] and Lewiner et al. [90] has finally provided triangulation patterns covering all possible topological configurations of trilinear functions, implicitly promising a crack-free surface. The topology of the level sets generated by trilinear interpolation has been recently studied by Carr and Snoeyink [17], and Carr and Max [15]. A discussion about these can be found in Section 3.3.2.

Verifiable Visualization. Many of the false steps in the route from the original MC algorithm to the recent homeomorphic solutions could have been avoided with a systematic procedure to verify the algorithms and the corresponding implementations.

Although the lack of verification of visualization techniques and the corresponding software implementations has been a long-term concern of the visualization community [51, 72], concrete proposals on verification are relatively recent. Etiene et al. [42] were among the first in scientific visualization to propose a practical verification framework for geometrical properties of isosurfacing. Their work is based on the method of manufactured solutions (MMS), a popular approach for assessing numerical software [3]. We are interested in *topological properties* of isosurfacing, and we also use MMS as a verification mechanism. As we will show in Section 3.5, our proposed technique discovered problems in popular software, supporting our assertion about the value of a broader culture of verification in scientific visualization.

There have been significant theoretical investigations in computational topology dealing with, for example, isosurface invariants, persistence, and stability [26, 34]. This body of work is concerned with how to define and compute topological properties of computational objects. We instead develop methods that stress topological properties of isosurfacing. These goals are complementary. Computational topology tools for data analysis might offer new properties which can be used for verification purposes, and verification tools can assess the correctness of the computational topology implementations. Although the mechanism we propose to compute topological invariants for piecewise smooth scalar fields is, to the best of our knowledge, novel (see Section 3.3.2), our primary goal is to present a method that developers can adapt to assess their own software.

3.2 Verifying Isosurface Topology

We now discuss strategies for verifying topological properties of isosurfacing techniques. We start by observing that simply stating the desired properties of the implementation is valuable. Consider a typical implementation of Marching Cubes. How would you debug it? Without a small set of desired properties, we are mostly limited to inspecting the output by explicitly exercising every case in the case table. The fifteen cases might not seem daunting, but what if we suspect a bug in symmetry reduction? We now have 256 cases to check. Even worse, what if the bug is in a combination of separate cases along neighboring cells? The verification would grow to be at least as complicated as the original algorithm, and we would just as likely make a mistake during the verification as we would in the implementation. Therefore, we need properties that are simple to state, easy to check, and good at catching bugs.

Simple example. Although the previously mentioned problem with Marching Cubes [96]

is well-known, it is not immediately clear what topological properties fail to hold. For example, "the output of Marching Cubes cannot contain boundary curves" is not one such property, for two reasons. First, some valid surfaces generated by Marching Cubes – such as with the simple 2^3 case – do contain boundaries. Second, many incorrect outputs might not contain any boundaries at all. The following might appear to be a good candidate property: "given a positive vertex v_0 and a negative vertex v_1, any path through the scalar field should intersect the isosurface an odd number of times." This property *does* capture the fact that the triangle mesh should separate interior vertices from exterior vertices and seems to isolate the problem with the cracks. Checking this property, on the other hand, and even stating it precisely, is problematic. Geometrical algorithms for intersection tests are notoriously brittle; for example, some paths might intersect the isosurface in degenerate ways. A more promising approach comes from noticing that any such separating isosurface has to be a piecewise-linear manifold, whose boundary must be a subset of the boundary of the grid. This directly suggests that "the output of Marching Cubes must be a piecewise-linear (PL) manifold whose boundaries are contained in the boundary of the grid." This property is simple to state and easy to test: the link of every interior vertex in a PL manifold is topologically a circle, and the link of every boundary vertex is a line. The term "consistency" has been used to describe problems with some algorithms [120]. In this work, we say that the output of an algorithm is *consistent* if it obeys the PL manifold property above. By generating arbitrary grids and extracting isosurfaces with arbitrary isovalues, the inconsistency of the original case table becomes mechanically checkable and instantly apparent. Some modifications to the basic Marching Cubes table, such as using Nielson and Hamann's asymptotic decider [122], result in consistent implementations, and the outputs pass the PL manifold checks (as we will show in Section 3.5).

The example we have presented above is a complete instance of the method of manufactured solutions. We identify a property that the results should obey, run the implementations on inputs, and test whether the resulting outputs respect the properties. In the next sections, we develop a verification method for algorithms to reproduce the topology of the level sets of trilinear interpolation [21, 94, 121], thus completely eliminating any ambiguity. In this work, we say the output is *correct* if it is homeomorphic to the corresponding level set of the scalar field. This correctness property is simple to state, but developing effective verification schemes that are powerful and simple to implement is more involved. We will turn to invariants of topological spaces, in particular to Betti numbers and the Euler characteristic, their relative strengths and weaknesses, and discuss how to robustly

check their values. Figure 3.1 shows our pipeline to assess topological correctness and also the chapter organization.

3.3 Mathematical Tools

This section describes the mathematical machinery used to derive the topology verification tools. More specifically, we provide a summary of the results we need from digital topology and stratified Morse theory. A detailed discussion on digital topology can be found in Stelldinger et al.'s paper [160], and Goresky and MacPherson give a comprehensive presentation of stratified Morse theory [53].

In Section 3.3.1, we describe a method, based on digital topology, that operates on manifold surfaces without boundaries and determines the Euler characteristic and Betti numbers of the level sets. A more general setting of surfaces with boundaries is handled with tools derived from stratified Morse theory, detailed in Section 3.3.2. The latter method

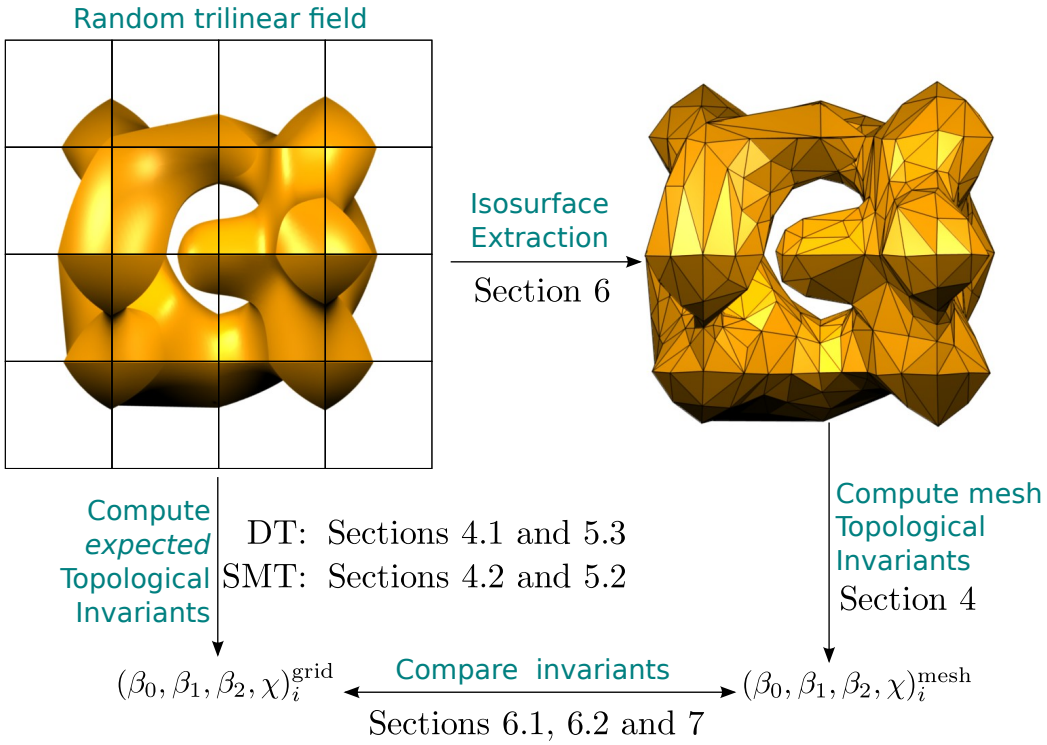


Figure 3.1. Overview of our topology verification pipeline. First step, we generate a random trilinear field and extract a random isosurface using the implementation under verification. We then compute the *expected* topological invariants from the trilinear field and compare them against the invariants obtained from the mesh. We provide two simple ways to compute topological invariants from a trilinear field based on digital topology (DT) or stratified Morse theory (SMT).

can only determine the Euler characteristic of the level set.

Let us start by recalling the definition and some properties of the Euler characteristic, which we denote by χ. For a compact 2-manifold $\mathcal{M}$, $\chi(\mathcal{M}) = V - E + F$, where V, E, and F are the number of vertices, edges, and faces of any finite cell decomposition of $\mathcal{M}$. If $\mathcal{M}$ is a connected orientable 2-manifold without boundary, $\chi(\mathcal{M}) = 2 - 2g(\mathcal{M})$, where $g(\mathcal{M})$ is the genus of $\mathcal{M}$. The Euler characteristic may also be written as $\chi(\mathcal{M}) = \sum_{i=0}^{n}(-1)^i\beta_i$, where β_i are the Betti numbers: the rank of the i-th homology group of $\mathcal{M}$. Intuitively, for 2-manifolds, β_0, β_1 and β_2 correspond to the number of connected components, holes and voids (regions of the space enclosed by the surface) respectively. If $\mathcal{M}$ has many distinct connected components, that is, $\mathcal{M} = \bigcup_{i=1}^{n}\mathcal{M}^i$ and $\mathcal{M}^i \bigcap \mathcal{M}^j = \emptyset$ for $i \neq j$ then $\chi(\mathcal{M}) = \sum_i^n \chi(\mathcal{M}^i)$. More details about Betti numbers, the Euler characteristic, and homology groups can be found in Edelsbrunner and Harer's text [34]. The Euler characteristic and the Betti numbers are topological invariants: two homeomorphic topological spaces will have the same Euler characteristic and Betti numbers whenever these are well-defined.

3.3.1 Digital topology

Let $\mathcal{G}$ be an $n \times n \times n$ cubic regular grid with a scalar $e(s)$ assigned to each vertex s of $\mathcal{G}$ and $t : \mathbb{R}^3 \to \mathbb{R}$ be the piecewise trilinear interpolation function in $\mathcal{G}$, that is, $t(x) = t_i(x)$, where t_i is the trilinear interpolant in the cubic cell c_i containing x. Given a scalar value α, the set of points satisfying $t(x) = \alpha$ is called the *isosurface* α of t. In what follows, $t(x) = \alpha$ will be considered a compact, orientable 2-manifold without boundary. We say that a cubic cell c_i of $\mathcal{G}$ is *unambiguous* if the following two conditions hold simultaneously:

1. any two vertices s_a and s_b in c_i for which $e(s_a) < \alpha$ and $e(s_b) < \alpha$ are connected by *negative edges*, i. e., a sequence of edges $s_a s_1, s_1 s_2, \ldots, s_k s_b$ in c_i whose vertices satisfy $e(s_i) < \alpha$ for $i = 1, \ldots, k$ and

2. any two vertices s_c and s_d in c_i for which $e(s_c) > \alpha$ and $e(s_d) > \alpha$ are connected by *positive edges*, i. e., a sequence of edges $s_c s_1, s_1 s_2, \ldots, s_l s_d$ in c_i whose vertices satisfy $e(s_i) > \alpha$ for $i = 1, \ldots, l$.

In other words, a cell is unambiguous if all positive vertices form a single connected component via the positive edges and, conversely, all negative vertices form a single connected component by negative edges [175]. If either property fails to hold, c_i is called *ambiguous*. The top row in Figure 3.2 shows all possible unambiguous cases.

The geometric dual of $\mathcal{G}$ is called the *voxel grid* associated with $\mathcal{G}$, denoted by $V_{\mathcal{G}}$. More specifically, each vertex s of $\mathcal{G}$ has a corresponding voxel v_s in $V_{\mathcal{G}}$, each edge of $\mathcal{G}$ corresponds

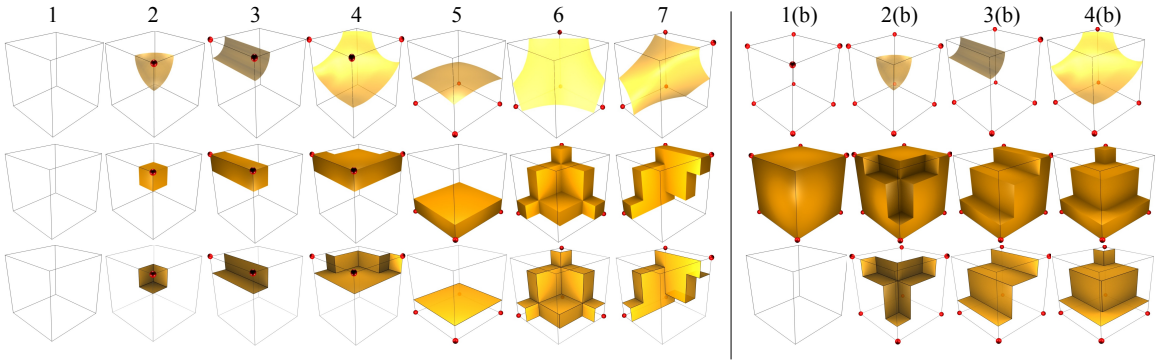


Figure 3.2. An illustration of the relation between unambiguous isosurfaces of trilinear interpolants and the corresponding digital surfaces. The top row shows all possible configurations of the intersection of $t = \alpha$ with a cube c_j for unambiguous configurations [94]. Each red dot s_i denotes a vertex with $e(s_i) < \alpha$. Each image on the top right is the complement $\bar{c}_i$ of cases 1 to 4 on the left (cases 5 to 7 were omitted because the complement is identical to the original cube up to symmetry). The middle row shows the volume reconstructed by Majority Interpolation (MI) for configurations 1 to 7 (left) and the complements (right) depicted in the top row. Bottom row shows the boundary of the volume reconstructed by the MI algorithm (The role of faces that intersect c_i is explained in the proof of Theorem 3.3.1). Notice that all surfaces in the top and bottom rows are topological disks. For each cube configuration, the boundary of each digital reconstruction (bottom row) has the same set of positive/negative connected components as the unambiguous configurations (top row).

to a face in $V_{\mathcal{G}}$ (and vice versa), and each cubic cell in $\mathcal{G}$ corresponds to a vertex in $V_{\mathcal{G}}$, as illustrated in Figure 3.3. Each voxel v_s can also be seen as the Voronoi cell associated with s. Scalars defined in the vertices of $\mathcal{G}$ can naturally be extended to voxels, thus ensuring a single scalar value $e(v_s)$ to each voxel v_s in $V_{\mathcal{G}}$ defined as $e(s) = e(v_s)$. As we shall show, the voxel grid structure plays an important role when using digital topology to compute topological invariants of a given isosurface. Before showing that relation, though, we need a few more definitions.

Denote by $\mathcal{G}'$ the $(2n-1) \times (2n-1) \times (2n-1)$ regular grid obtained from a refinement of $\mathcal{G}$. Vertices of $\mathcal{G}'$ can be grouped in four distinct sets, denoted by O, F, E, C. The set O contains the vertices of $\mathcal{G}'$ that are also vertices of $\mathcal{G}$. The sets F and E contain the vertices of $\mathcal{G}'$ lying on the center of faces and edges of the voxel grid $V_{\mathcal{G}}$, respectively. Finally, C contains all vertices of $V_{\mathcal{G}}$. Figure 3.3 illustrates these sets.

Consider now the voxel grid $V_{\mathcal{G}'}$ dual to the refined grid $\mathcal{G}'$. Given a scalar value α, the *digital object* $\mathcal{O}_\alpha$ is the subset of voxels v in $V_{\mathcal{G}'}$ such that $v \in \mathcal{O}_\alpha$ if at least one of the criteria below are satisfied:

- $v \in O$ and $e(v) \leq \alpha$

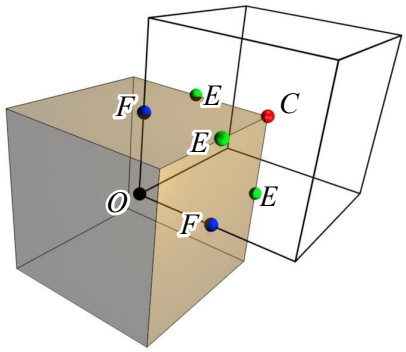


Figure 3.3. The four distinct groups of vertices O, F, E, C, are depicted as black, blue, green, and red points. They are the "Old", "Face", "Edge," and "Corner" points of a voxel region $V_{\mathcal{G}}$ (semitransparent cube) respectively. For the sake of clarity, we only show a few points.

- $v \in F$ and both neighbors of v in O have scalars less than (or equal to) α
- $v \in E$ and at least 4 of the 8 neighbors of v in $O \cup F$ have scalars less than (or equal) α
- $v \in C$ and at least 12 of the 26 neighbors of v in $O \cup F \cup E$ have scalars less than (or equal) α

The description above is called Majority Interpolation (MI) (Algorithm 2), and it allows us to compute the voxels that belong to a digital object $\mathcal{O}_\alpha$. The middle row of Figure 3.2 shows all possible cases for voxels picked by the MI algorithm (notice the correspondence with the top row of the same figure).

The importance of $\mathcal{O}_\alpha$ is two-fold. First, the boundary surface of the union of the voxels in $\mathcal{O}_\alpha$, denoted by $\partial\mathcal{O}_\alpha$ and called a *digital surface*, is a 2-manifold (See the proof by Stelldinger et al. [160]). Second, the genus of $\partial\mathcal{O}_\alpha$ can be computed directly from $\mathcal{O}_\alpha$ using the algorithm proposed by Chen and Rong [20] (Algorithm 5). As the connected components of $\mathcal{O}_\alpha$ can also be easily computed and isolated, one can calculate the Euler characteristic of each connected component of $\mathcal{O}_\alpha$ from the formula $\chi = 2 - 2g$ and thus β_0, β_1, and β_2.

The voxel grid $V_{\mathcal{G}'}$ described above allows us to compute topological invariants for any digital surface $\partial\mathcal{O}_\alpha$. However, we so far do not have any result relating $\partial\mathcal{O}_\alpha$ to the isosurface $t(x) = \alpha$. The next theorem provides the connection.

Theorem 3.3.1. *Let $\mathcal{G}$ be an $n \times n \times n$ rectilinear grid with scalars associated with each vertex of $\mathcal{G}$ and t be the piecewise trilinear function defined on $\mathcal{G}$, such that the isosurface*

Algorithm 2 Voxel selection using Majority Interpolation (MI).

MajorityInterpolation($\mathcal{G}, \alpha$)

```
▷ Let O, F, E and C be the subset of vertices
  in G' as described in subsection 3.3.1.
▷ Let N(s, ⋆) be the set of neighbors of s ∈ G' in the
  set ⋆, where ⋆ = {O, F, E, C}, with associate scalar
  less than α
for s ∈ G'
    do if s ∈ O or
        s ∈ F and |N(s, O)| = 2 or
        s ∈ E and |N(s, O) + N(s, F)| ⩾ 4 or
        s ∈ C and |N(s, O) + N(s, F) + N(s, E)| ⩾ 12
            then Select voxel v_s
return O_α
```

$t(x) = \alpha$ is a 2-manifold without boundary. If no cubic cell of $\mathcal{G}$ is ambiguous with respect to $t(x) = \alpha$, then $\partial\mathcal{O}_\alpha$ is homeomorphic to the isosurface $t(x) = \alpha$.

Proof: Given a cube $c_i \subset \mathcal{G}$ and an isosurface $t = \{x \mid t(x) = \alpha\}$, let $t_i = t \cap c_i$. Similarly, denote

$$\partial\mathcal{O}_i = cl_{\mathbb{R}^3}\left((\partial\mathcal{O}_\alpha \cap c_i) - \partial c_i\right),$$

where $cl_{\mathbb{R}^3}$ denotes the closure operator. We note that $\partial\mathcal{O}_i$ is a 2-manifold for all i [143, 160]. There are two main parts to the proof presented here. For each i,

1. the 2-manifolds t_i and $\partial\mathcal{O}_i$ are homeomorphic; and

2. both t_i and $\partial\mathcal{O}_i$ cut the same edges and faces of c_i.

Since t is trilinear, no level-set of t can intersect an edge more than once. Hence, if c_i is not ambiguous, t_i is exactly one of the cases 1 to 7 in the top row of Figure 3.2 [94], either a topological disk or the empty set. Each case in the top row of Figure 3.2 is the unambiguous input for the MI algorithm to produce the voxel reconstruction shown in the middle row, where the boundaries of each of these voxel reconstructions are shown in the bottom row. By inspection, we can verify that the boundary of the digital reconstruction $\partial\mathcal{O}_i$ (bottom row of Figure 3.2) is also a disk for all possible unambiguous cases and complement cases. Hence, for each i, the 2-manifolds $\partial\mathcal{O}_i$ and t_i are homeomorphic. Then, for each i, both $\partial\mathcal{O}_i$ and t_i cut the same set of edges and faces of c_i. Again, we can verify this for all possible i by inspecting the top and bottom rows in Figure 3.2, respectively. Finally, we apply the

Pasting Lemma [116] across neighboring surfaces $\partial\mathcal{O}_i$ and $\partial\mathcal{O}_j$ in order to establish the homeomorphism between $\partial\mathcal{O}_\alpha$ and t. □

This proof provides a main ingredient for the verification method in Section 3.4. Crucially, we will show how to manufacture a complex solution that unambiguously crosses every cubic cell of the grid. Since we have shown the conditions for which the digital surfaces and the level sets are homeomorphic, any topological invariant will have to be the same for both surfaces.

3.3.2 Stratified Morse Theory

The mathematical developments presented above allow us to compute the Betti numbers of any isosurface of the piecewise trilinear interpolant. However, they require isosurfaces without boundaries. In this section, we provide a mechanism to compute the Euler characteristic of any regular isosurface of the piecewise trilinear interpolant through an analysis based on critical points, which can be used to verify properties of isosurfaces with boundary components. We will use some basic machinery from stratified Morse theory (SMT), following the presentation of Goresky and MacPherson's monograph [53].

Let f for now be a smooth function with isolated critical points p, where $\nabla f(p) = 0$. From classical Morse theory, the topology of two isosurfaces $f(x) = \alpha$ and $f(x) = \alpha+\epsilon$ differs only if the interval $[\alpha, \alpha+\epsilon]$ contains a critical value ($f(p)$ is a critical value iff p is a critical point). Moreover, if ε_p is a small neighborhood around p and $L^-(p)$ and $L^+(p)$ are the subset of points on the boundary of ε_p satisfying $f(x) < f(p)$ and $f(x) > f(p)$ respectively, then the topological change from the isosurface $f(x) = f(p)-\epsilon$ to $f(x) = f(p)+\epsilon$ is characterized by removing $L^-(p)$ and attaching $L^+(p)$. Thus, changes in the Euler characteristic, denoted by $\Delta\chi(p)$, are given by:

$$\Delta\chi(p) = \chi(L^+(p)) - \chi(L^-(p)). \tag{3.1}$$

For a smooth function f, the number of negative eigenvalues of the Hessian matrix determines the index of a critical point p, and the four cases give the following values for $\chi(L^-(p))$ and $\chi(L^+(p))$:

	min	saddle-1	saddle-2	max
$\chi(L^-(p))$	0	2	0	2
$\chi(L^+(p))$	2	0	2	0

The above formulation is straightforward but unfortunately cannot be directly applied to functions appearing in either piecewise trilinear interpolations or isosurfaces with boundary, both of which appear in some of the isosurfacing algorithms with guaranteed topology.

Trilinear interpolants are not smooth across the faces of grid cells, so the gradient is not well-defined there. Identifying the critical points using smooth Morse theory is then problematic. Although arguments based on smooth Morse theory have appeared in the literature [178], there are complications. For example, the scalar field in a node of the regular grid might not have *any* partial derivatives. Although one can still argue about the intuitive concepts of minima and maxima around a non-differentiable point, configurations such as saddles are more problematic, since their topological behavior is different depending on whether they are on the boundary of the domain. It is important, then, to have a mathematical tool which makes predictions regardless of the types of configurations, and SMT is one such theory.

Intuitively, a *stratification* is a partition of a piecewise-smooth manifold such that each subset, called a *stratum*, is either a set of discrete points or has a smooth structure. In a regular grid with cubic cells, the stratification we propose will be formed by four sets (the strata), each one a (possibly disconnected) manifold. The *vertex set* contains all vertices of the grid. The *edge set* contains all edge interiors, the *face set* contains all face interiors, and the *cell set* contains all cube interiors. We illustrate the concept for the 2D case in Figure 3.4. The important property of the strata is that the level sets of f restricted to each stratum are smooth (or lack any differential structure, as in the vertex-set). In SMT, one applies standard Morse theory on each stratum, and then combines the partial results appropriately.

The set of points with zero gradient (computed on each stratum), which SMT assumes to be isolated, are called the *critical points* of the stratified Morse function. In addition, every point in the vertex set is considered critical as well. One major difference between SMT and the smooth theory is that some critical points do not actually change the topology of the level sets. This is why considering all grid vertices as critical does not introduce any practical problems: most grid vertices of typical scalar fields will be *virtual critical points*, *i.e.*, points which do not change the Euler characteristic of the surface. Carr and Snoeyink use a related concept (which they call "potential critical points") in their state-machine description of the topology of interpolants [17].

Let $\mathcal{M}$ be the stratified grid described above. It can be shown that if p is a point in a d-dimensional stratum of $\mathcal{M}$, it is always possible to find a $(3-d)$-dimensional submanifold of $\mathcal{M}$ (which might straddle many strata) that meets transversely the stratum containing p, and whose intersection consists of only p (one way to think of this $(3-d)$-manifold is as a "topological orthogonal complement"). In this context, we can

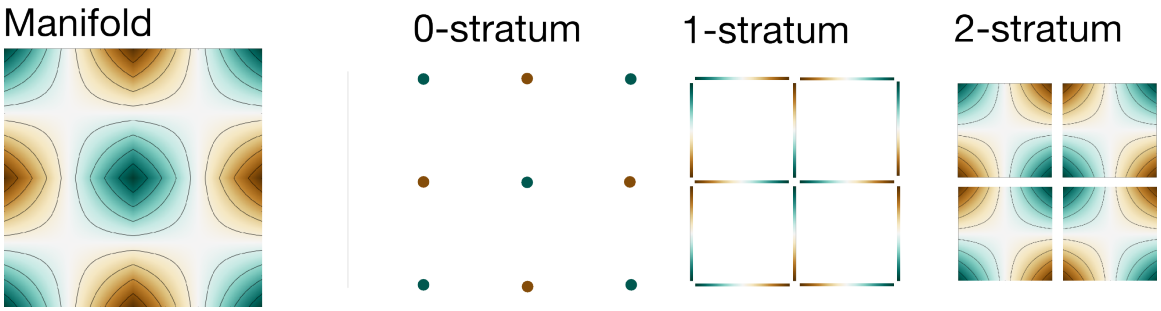
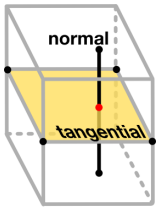


Figure 3.4. An illustration of a piecewise-smooth immersed 2-manifold. The colormap illustrates the value of each point of the scalar field. Notice that although the manifold itself is not everywhere differentiable, each stratum is itself an open manifold that is differentiable.

define a small neighborhood $T_\varepsilon(p)$ in the strata containing p and the *lower tangential* link $T_L^-(p)$ as the set of points in the boundary of $T_\varepsilon(p)$ with scalar values less than that in p. Similarly we can define the *upper tangential* link $T_L^+(p)$ as the set of points in the boundary of $T_\varepsilon(p)$ with scalar value higher than that at p. *Lower normal* $N_L^-(p)$ and *upper normal* $N_L^+(p)$ links are analogous notions, but the lower and upper links are taken to be subsets of $N_\varepsilon(p)$, itself a subset of the $(3-d)$-dimensional submanifold transverse to the stratum of p going through p. The definitions above are needed in order to define the *lower stratified link* and *upper stratified link*, as follows: given $T_\varepsilon(p)$, $T_L^-(p)$, $N_\varepsilon(p)$ and $N_L^-(p)$, the *lower stratified Morse link* (and similarly for upper stratified link) is given by



$$L^-(p) = (T_\varepsilon(p) \times N_L^-(p)) \cup (N_\varepsilon(p) \times T_L^-(p)). \tag{3.2}$$

These definitions allow us to classify critical points even in the non-smooth scenario. They let us compute topological changes with the same methodology used in the smooth case. In other words, when a scalar value α crosses a critical value α_p in a critical point p, the topological change in the isosurface is characterized by removing $L^-(p)$ and attaching $L^+(p)$, affecting the Euler characteristic as defined in Equation 3.1.

The remaining problem is how to determine $\chi(L^-(p))$ and $\chi(L^+(p))$. Recalling that $\chi(A \cup B) = \chi(A) + \chi(B) - \chi(A \cap B)$, $\chi(A \times B) = \chi(A)\chi(B)$, and $\chi(T_\varepsilon) = \chi(N_\varepsilon) = 1$ (we are omitting the point p) we have:

$$\begin{aligned} \chi(L^-) &= \chi(T_\varepsilon \times N_L^- \cup N_\varepsilon \times T_L^-) \\ &= \chi(N_L^-) + \chi(T_L^-) - \chi(T_\varepsilon \times N_L^- \cap N_\varepsilon \times T_L^-) \end{aligned} \tag{3.3}$$

Now, we can define $T_\varepsilon = T_L^- \cup T_r$, $T_L^- \cap T_r = \emptyset$ and similarly for N_ε and N_L^-. Then, expand the partitions and products, and distribute the intersections around the unions, noticing all

but one of intersections will be empty:

$$\begin{aligned} T_\varepsilon \times N_L^- \cap N_\varepsilon \times T_L^- &= ((T_r \cup T_L^-) \times N_L^-) \cap ((N_r \cup N_L^-) \times T_L^-) \\ &= ((T_r \times N_L^-) \cup (T_L^- \times N_L^-)) \cap \\ &\quad ((N_r \times T_L^-) \cup (N_L^- \times T_L^-)) \\ &= N_L^- \times T_L^- \end{aligned}$$

Therefore:

$$\begin{aligned} \chi(T_\varepsilon \times N_L^- \cap N_\varepsilon \times T_L^-) &= \chi(N_L^- \times T_L^-) \\ &= \chi(N_L^-)\chi(T_L^-) \end{aligned}$$

which gives the final result

$$\chi(L^-) = \chi(N_L^-) + \chi(T_L^-) - \chi(N_L^-)\chi(T_L^-). \tag{3.4}$$

The same result is valid for $\chi(L^+)$, if we replace the superscript '$-$' by '$+$' in Equation 3.4. If T_L^- or T_L^+ are one-dimensional, then we are done. If not, then we can recursively apply the same equation to T_L^- and T_L^+ and look at progressively lower-dimensional strata until we reach $T_\varepsilon(p)$ and $N_\varepsilon(p)$ given by 1-disks. The lower and upper links for these 1-disks will always be discrete spaces with zero, one, or two points, for which χ is simply the cardinality of the set.

In some cases, the Euler characteristic of the lower and upper link might be equal. Then, $\chi(L^-(p)) = \chi(L^+(p))$, and $\Delta\chi(p) = 0$. These cases correspond to the virtual critical points mentioned above. Critical points in the interior of cubic cells are handled by the smooth theory, since in that case the normal Morse data is 0-dimensional. This implies that the link will be an empty set with Euler characteristic zero. So, by Equation 3.4, $\chi(L^-) = \chi(T_L^-)$. Because the restriction of the scalar field to a grid edge is a linear function, no critical point can appear there. As a result, the new cases are critical points occurring at vertices or in the interior of faces of the grid. For a critical point p in a vertex, stratification can be carried out recursively, using the edges of the cubes meeting in p as tangential and normal submanifolds. Denoting by n_{l1}, n_{l2}, n_{l3} the number of vertices adjacent to p with scalar value less than that of p in each Cartesian coordinate direction, Equation (3.4) gives:

$$\chi(L^-(p)) = n_{l1} + n_{l2} + n_{l3} - n_{l1}(n_{l2} + n_{l3}) \tag{3.5}$$

$\chi(L^+(p))$ can be computed similarly, but considering the number of neighbors of p in each Cartesian direction with scalars higher than that of p.

If p is a critical point lying in a face r of a cube, we consider the face itself as the tangential submanifold and the line segment $r^{\perp}$ orthogonal to r through p the normal submanifold. Recursively, the tangential submanifold can be further stratified in two 1-disks (tangential and normal). Denote by n_l the number of ends of $r^{\perp}$ with scalar value less than that of p. Also, recalling that the critical point lying in the face r is necessarily a saddle, thus having two face corners with scalar values less and two higher than that of p, Equation (3.4) gives:

$$\chi(L^{-}(p)) = n_l + 2 - 2\,n_l \tag{3.6}$$

Analogously, we can compute $\chi(L^{+}(p)) = n_u + 2 - 2\,n_u$ where n_u is the number of ends of $r^{\perp}$ with scalar value higher than that of p.

A similar analysis can be be carried out for every type of critical point, regardless of whether the point belongs to the interior of a grid cell (and so would yield equally well to a smooth Morse theory analysis), an interior face, a boundary face, or a vertex of any type. The Euler characteristic χ_α of any isosurface with isovalue α is simply given as:

$$\chi_\alpha = \sum_{p_i \in C_\alpha} \Delta\chi(p_i) \tag{3.7}$$

where C_α is the set of critical points with critical values less than α.

It is worth mentioning once again that, to the best of our knowledge, no other work has presented a scheme which provides such a simple mechanism for computing the Euler characteristic of level sets of piecewise-smooth trilinear functions. Compare, for example, the case analyses and state machines performed separately by Nielson [121], by Carr and Snoeyink [17], and by Carr and Max [15]. In contrast, we can recover an (admittedly weaker) topological invariant by a much simpler argument. In addition, this argument already generalizes (trivially because of the stratification argument) to arbitrary dimensions, unlike the other arguments in the literature.

3.4 Manufactured Solution Pipeline

We now put the pieces together and build a pipeline for topology verification using the results presented in Section 3.3. In the following sections, the procedure called IsosurfacingMC refers to the isosurface extraction technique under verification. InvariantFromMesh computes topological invariants of a simplicial complex.

3.4.1 Consistency

As previously mentioned, MC-like algorithms which use disambiguation techniques are expected to generate PL manifold isosurfaces no matter how complex the function sampled

in the vertices of the regular grid. In order to stress the consistency test, we generate a random scalar field with values in the interval $[-1, 1]$ and extract the isosurface with isovalue $\alpha = 0$ (which is all but guaranteed not to be a critical value) using a given isosurfacing technique, subjecting the resulting triangle mesh to the consistency verification. This process is repeated a large number of times, and if the implementation fails to produce PL manifolds for all cases, then the counterexample provides a documented starting point for debugging. If it passes the tests, we consider the implementation verified.

3.4.2 Verification using Stratified Morse Theory

We can use the formulation described in Section 3.3.2 to verify isosurfacing programs which promise to match the topology of the trilinear interpolant. The SMT-based verification procedure is summarized in Algorithm 3. The algorithm has four main steps. A random scalar field with node values in the interval $[-1, 1]$ is initially created. Representing the trilinear interpolation in a grid cell by $f(x, y, z) = axyz+bxy+cxz+dyz+ex+fy+gz+h$, the internal critical points are given by:

$$\begin{aligned} t_x &= (d\Delta_x \pm \sqrt{\Delta_x\Delta_y\Delta_z})/(a\Delta_x) \\ t_y &= (c\Delta_y \pm \sqrt{\Delta_x\Delta_y\Delta_z})/(a\Delta_y) \\ t_z &= (b\Delta_z \pm \sqrt{\Delta_x\Delta_y\Delta_z})/(a\Delta_z), \end{aligned}$$

where $\Delta_x = bc - ae$, $\Delta_y = bd - af$, and $\Delta_z = cd - ag$ [126]. Critical points on faces of the cubes are found by setting x, y or z to either 0 or 1, and solving the quadratic equation. If the solutions lie outside the unit cube $[0, 1]^3$, they are not considered critical points, since they lie outside the domain of the cell. The scalar field is regenerated if any degenerate critical point is detected (these can happen if either the random values in a cubic cell have, by chance, the same value or when Δ_x, Δ_y or Δ_z are zero). In order to avoid numerical instabilities, we also regenerate the scalar field locally if any internal critical point lies too close to the border of the domain (that is, to an edge or to a face of the cube).

The third step computes the Euler characteristic of a set of isosurfaces with random isovalues in the interval $[-1, 1]$ using the theory previously described, jointly with Equation 3.7. In the final step, the triangle mesh M approximating the isosurfaces is extracted using the algorithm under verification, and $\chi(M) = V(M)-E(M)+F(M)$, where $V(M), E(M)$, and $F(M)$ are the number of vertices, edges, and triangles. If the Euler characteristic computed from the mesh does not match the one calculated via Equation 3.7, the verification fails. We carry out the process a number of times, and implementations that pass the tests are less likely to contain bugs.

Algorithm 3 Overview of the method of manufactured solutions (MMS) using stratified Morse theory. InvariantFromCPs is computed using Equation 3.7. The method either fails to match the expected topology, in which case $\mathcal{G}$ is provided as a counterexample, or succeeds otherwise.

```
MMS-SMT(G)
 1  ▷ Let the input G be n × n × n rectilinear grid
 2  for i ← 1 to #tests
 3       do G ← randomly sampled n × n × n grid
 4          CPs ← ComputeCriticalPoints(G)
 5          if p ∈ CPs is degenerate or
 6          p is an internal saddle close to edges or faces
 7            then GoTo 3
 8            else K ← Isosurfacing(G)
 9                 (χ^v)_i ← InvariantFromCPs(G)
10                 (χ^k)_i ← InvariantFromMesh(K)
11          Compare (χ^v)_i and (χ^k)_i
```

3.4.3 Verification using Digital Topology

Algorithm 4 shows the verification pipeline using the MI algorithm, and Figure 3.5 depicts the refinement process. Once again a random scalar field, with potentially many ambiguous cubes, is initially generated in the vertices of a grid $\mathcal{G}$. The algorithm illustrated in Algorithm 4 is applied to refine $\mathcal{G}$ so as to generate a new grid $\tilde{\mathcal{G}}$ which does not have ambiguous cells. If the maximum number of refinement is reached and ambiguous cells still remain, then the process is restarted from scratch. Notice that cube subdivision does not need to be uniform. For instance, each cube may be refined using a randomly placed new node point or using t_i's critical points, and the result of the verification process still holds. This is because Theorem 3.3.1 only requires c_i to be unambiguous. For simplicity, in this work we refine $\mathcal{G}$ uniformly doubling the grid resolution in each dimension.

Scalars are assigned to the new vertices of $\tilde{\mathcal{G}}$ (the ones not in $\mathcal{G}$) by trilinearly interpolating from scalars in $\mathcal{G}$, thus ensuring that $\mathcal{G}$ and $\tilde{\mathcal{G}}$ have exactly the same scalar field [121]. As all cubic cells in $\tilde{\mathcal{G}}$ are unambiguous, Theorem 3.3.1 guarantees the topology of the digital surface $\partial\mathcal{O}_\alpha$ obtained from $\tilde{\mathcal{G}}$ is equivalent to that of $t(x) = \alpha$. Algorithm InvariantFromDS computes topological invariants of $\partial\mathcal{O}_\alpha$ using the scheme discussed in Section 3.3.1. In this context, InvariantFromDS is the algorithm illustrated in Algorithm 5. Surfaces with boundary are avoided by assigning the scalar value 1 to every vertex in the boundary of $\mathcal{G}$.

Algorithm 4 Overview of the method of manufactured solutions (MMS) using digital topology. The method either fails to match the expected topology, in which case $\mathcal{G}$ is provided as a counterexample, or succeeds otherwise.

MMS-DS($\mathcal{G}$)

```
 1  ▷ Let the input G be a n × n × n rectilinear grid
 2  for i ← 1 to #tests
 3        do G ← randomly sampled n × n × n grid
 4           G̃ ← RefineAndResample(G)
 5           if G̃ has ambiguous cubes
 6              then GoTo 3
 7           O ← MajorityInterpolation(G̃)
 8           K ← Isosurfacing(G)
 9           (β0^v, β1^v, β2^v)_i ← InvariantFromDS(∂O)
10           (β0^k, β1^k, β2^k)_i ← InvariantFromMesh(K)
11           Compare (β0^v, β1^v, β2^v)_i and (β0^k, β1^k, β2^k)_i
```

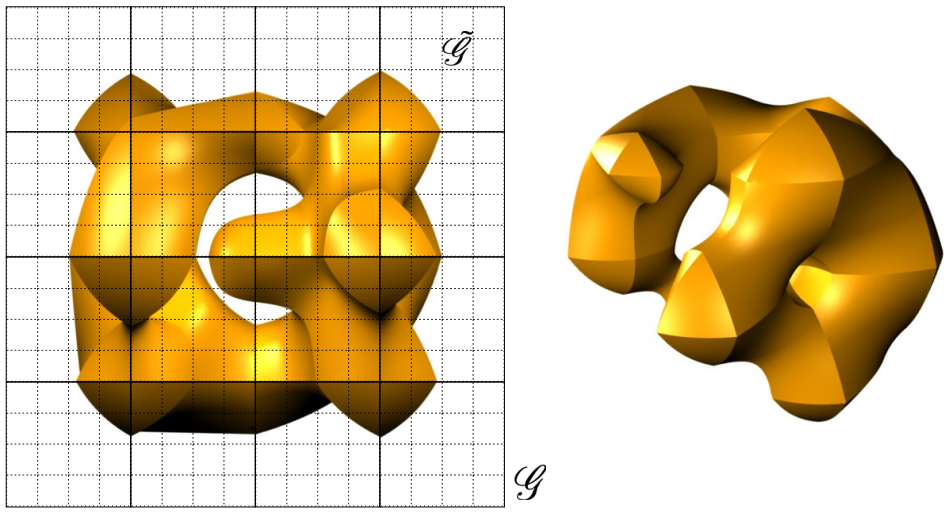


Figure 3.5. Our manufactured solution is given by $t(x) = \alpha$. $\mathcal{G}$ is depicted in solid lines while $\tilde{\mathcal{G}}$ is shown in dashed lines. $\tilde{\mathcal{G}}$ is a uniform subdivision of $\mathcal{G}$. The trilinear surfaces t_i are defined for each cube $c_i \in \mathcal{G}$ and resampled in $c'_j \in \tilde{\mathcal{G}}$. The cubes in the center of $\mathcal{G}$ have four maxima each (left) and thus induce complicated topology. The final isosurface may have several tunnels and/or connected components even for coarse $\mathcal{G}$ (right).

3.5 Experimental Results

In this section, we present the results of applying our topology verification methodology to a number of different isosurfacing techniques, three of them with topological guarantees with respect to trilinear interpolant. Specifically, the techniques are:

VTKMC [151] is the Visualization Toolkit (VTK) implementation of the Marching

Cubes algorithm with the implicit disambiguation scheme proposed by Montani et al. [113]. Essentially, it separates positive vertices when a face saddle appears and assumes no tunnels exist inside a cube. The proposed scheme is topologically consistent, but it does not reproduce the topology of the trilinear interpolant.

Marching Cubes with Edge Transformations or MACET [31] is a Marching Cubes-based technique designed to generate triangle meshes with good quality. Quality is reached by displacing active edges of the grid (edges intersected by the isosurface), both in normal and tangential direction toward avoiding "sliver" intersections. Macet does not reproduce the topology of the trilinear interpolant.

AFRONT [149] is an advancing-front method for isosurface extraction, remeshing, and triangulation of point sets. It works by advancing triangles over an implicit surface. A sizing function that takes curvature into account is used to adapt the triangle mesh to features of the surface. AFRONT uses cubic spline reconstruction kernels to construct the scalar field from a regular grid. The algorithm produces high-quality triangle meshes with bounded Hausdorff error. As occurred with the VTK and Macet implementations, Afront produces consistent surfaces but, as expected, the results do not match the trilinear interpolant.

MATLAB® [99] is a high-level language for building codes that requires intensive numerical computation. It has a number of features and among them an isosurface extraction routine for volume data visualization. Unfortunately, MATLAB documentation does not offer information on the particularities of the implemented isosurface extraction technique (e.g., Marching Cubes, Delaunay-based, etc; consistent or correct).

SNAPMC [137] is a Marching Cubes variant which produces high-quality triangle meshes from regular grids. The central idea is to extend the original lookup table to account for cases where the isosurface passes exactly through the grid nodes. Specifically, a user-controlled parameter dictates maximum distance for "snapping" the isosurface into the grid node. The

Algorithm 5 A simple formula for genus computation.

GENUSFROMDS($\partial\mathcal{O}_\alpha$)
1 $\triangleright$ Let $\partial\mathcal{O}_\alpha$ be a 2-manifold without boundary
2 $\triangleright$ Let $|\mathcal{N}_i|$ be the number of surface points with exactly i neighbors.
3 $\triangleright$ Let g be the surface genus
4 $g = 1 + (|\mathcal{N}_5| + 2|\mathcal{N}_6| - |\mathcal{N}_3|)/8$
5 **return** g

authors report an improvement in the minimum triangle angle when compared to previous techniques.

MC33 was introduced by Chernyaev [21] to solve ambiguities in the original MC. It extends Marching Cubes table from 15 to 33 cases to account for ambiguous cases and to reproduce the topology of the trilinear interpolant inside each cube. The original table was later modified to remove two redundant cases which leads to 31 unique configurations. Chernyaev's MC solves face ambiguity using Nielsen and Hamann's [122] asymptotic decider and internal ambiguity by evaluating the bilinear function over a plane parallel to a face. Additional points may be inserted to reproduce some configuration requiring subvoxel accuracy. We use Lewiner et al.'s implementation [90] of Chernyaev's algorithm.

DELISO [30] is a Delaunay-based approach for isosurface extraction. It uses the intersection of the 3D Voronoi diagram and the desired surface to define a restricted Delaunay triangulation. Moreover, it builds the restricted Delaunay triangulation without having to compute the whole 3D Voronoi structure. DELISO has theoretical guarantees of homeomorphism and mesh quality.

MCFLOW is a proof-of-concept implementation of the algorithm described in Scheidegger et al. [147]. It works by successive cube subdivision until it has a *simple edge flow*. A cube has a simple edge flow if it has only one *minima* and one *maxima*. A vertex $s \in c_i$ is a minimum if all vertices $s_j \in c_i$ connected to it has $t(s_j) > t(s_i)$. Similarly, a vertex is a maximum if $t(s_j) < t(s_i)$ for every neighbor vertex j. This property guarantees that the Marching Cubes method will generate a triangle mesh homeomorphic to the isosurface. After subdivision, the surfaces must be attached back together. The final mesh is topologically correct with respect to the trilinear interpolant.

We believe that the implementation of any of these algorithms in full detail is non-trivial. The results reported in the following section support this statement. They show that coding isosurfacing algorithms is complex and error-prone, and they reinforce the need for robust verification mechanisms. In what follows, we say that a *mismatch* occurs when invariants computed from a verification procedure disagree with the invariants computed from the isosurfacing technique. A mismatch does not necessarily mean an implementation is incorrect, as we shall see later in this section. After discussions with the developers, however, we did find that there were bugs in some of the implementations.

3.5.1 Topology consistency

All implementations were subject to the consistency test (Section 3.4.1), resulting in the outputs reported in the first column of Table 3.1. We observed mismatches for DELISO,

SnapMC (with non-zero snap value), and Matlab implementations. Now, we detail these results.

3.5.1.1 DelIso

We analyzed 50 cases where DelIso's output mismatched the ground truth produced by MMS, and we found that: 1) 28 cases had incorrect hole(s) in the mesh, 2) 15 cases had missing triangle(s), and 3) 7 cases had duplicated vertices. These cases are illustrated in Figure 3.7. The first problem is possibly due to the non-smooth nature of the piecewise trilinear interpolant, since in all 28 cases the holes appeared in the faces of the cubic grid. It is important to recall that DelIso is designed to reproduce the topology of the trilinear interpolant inside each grid cube, but the underlying algorithm requires the isosurface to be C^2 continuous everywhere, which does not hold for the piecewise trilinear isosurface. In practice, real world datasets such as medical images may induce "smoother" piecewise trilinear fields when compared to the extreme stressing from the random field, which should reduce the incidence of such cases. Missing triangles, however, occurred in the interior of cubic cells where the trilinear surface is smooth. Those problems deserve a deeper analysis, as one cannot say beforehand if the mismatches are caused by problems in the code or numerical instability associated with the initial sampling, ray-surface intersection, and the 3D Delaunay triangulation construction.

3.5.1.2 SnapMC

Table 3.1 shows that SnapMC with non-zero snap value causes the mesh to be topologically inconsistent (Figure 3.9(a)) in more the 50% of the performed tests. The reason for this behavior is in the heart of the technique: the snapping process causes geometrically close vertices to be merged together which may eliminate connected components, or loops, join connected components or even create non-manifold surfaces. This is why there was an increase in the number of mismatches when compared with SnapMC with zero snap value. Since non-manifold meshes are not desirable in many applications, the authors suggest a post-processing for fixing these topological issues, although no implementation or algorithm for this post-processing is provided.

3.5.1.3 Matlab

Matlab documentation does not specify the properties of the implemented isosurface extraction technique. Consequently, it becomes hard to justify the results for the high number of mismatches we see in Table 3.1. For instance, Figure 3.9(b) shows an example

of a non-manifold mesh extracted using MATLAB. In that figure, the two highlighted edges have more than two faces connected to them and the faces between these edges are coplanar. Since we do not have enough information to explain this behavior, this might be the actual expected behavior or an unexpected side effect. An advantage of our tests is the record of the observed behavior of mesh topologies generated by MATLAB.

3.5.1.4 MACET

In our first tests, MACET failed in all consistency tests for a $5 \times 5 \times 5$ grid. An inspection in the code revealed that the layer of cells in the boundary of the grid has not been traversed. Once that bug was fixed, MACET started to produce PL manifold meshes and was successful in the consistency test, as shown in Table 3.1.

3.5.2 Topology correctness

The verification tests described in Section 3.4.2 and 3.4.3 were applied to all algorithms, although only MC33, DELISO, and MCFLOW were expected to generate meshes with the same topology of the trilinear interpolant. Our tests consisted of one thousand random fields generated in a rectilinear $5 \times 5 \times 5$ grid $\mathcal{G}$. The verification test using Digital Surfaces demanded a compact, orientable, 2-manifold without boundary, so we set scalars equal 1 for grid vertices in the boundary of the grid. As stratified Morse theory supports surfaces with boundary, no special treatment was employed in the boundary of $\mathcal{G}$. We decided to run these tests using all algorithms for completeness and also for testing the tightness of the theory, which says that if the algorithms do not preserve the topology of the trilinear interpolant, a mismatch should occur. Interestingly, with this test, we were able to find another code mistake in MACET that prevented it from terminating safely when the SMT procedure was applied. For all non topology-preserving algorithms, there was a high number of mismatches as expected.

One might think that the algorithms described in Algorithms 3 and 4 do not cover all possible topology configurations because some scalar fields are eventually discarded (lines 7 and 6 respectively). This could happen due to the presence of ambiguous cells after refining the input grid to the maximum tolerance (digital topology test) or critical points falling too close to edges/faces of the cubic cells (SMT test). However, we can ensure that all possible configurations for the trilinear interpolation were considered in the tests. Figure 3.6 shows the incidence of each possible configuration (including all ambiguous cases) for the trilinear interpolation in the generated random fields. Dark bars correspond to the number of times a specific case happens in the random field, and the light bars show how many of those cases

are accepted by our verification methodology, that is, the random field is not discarded. Notice that no significant differences can be observed, implying that our rejection-sampling method does not bias the case frequencies.

Some configurations, such as 13 or 0, have low incidence rates and therefore might not be sufficiently stressed during verification. While the trivial case 0 does not pose a challenge for topology-preserving implementations, configuration 13 has 6 subcases whose level-sets are fairly complicated [94, 121]. Fortunately, we can build random fields in a convenient fashion by forcing a few cubes to represent a particular instance of the table, such as case 13, which produces more focused tests.

Table 3.1 shows statistics for all implementations. For MC33, the tests revealed a problem with configuration 4, 6, and 13 of the table (ambiguous cases). Figure 3.8 shows the obtained and expected tiles for a cube. Contacting the author, we found that one of the mismatches was due to a mistake when coding configuration 13 of the MC table. A non-obvious algorithm detail that is not discussed in either Chernyaev's or Lewiner's work is the problem of orientation in some of the cube configurations [88]. The case 13.5.2 shown in Figure 3.8 (right) is an example of one such configuration, where an additional criterion is required to decide the tunnel orientation that is lacking in the original implementation of MC33. This problem was easily detected by our framework, because the orientation changes the mesh invariants, and a mismatch occurs.

DelIso presented a high percentage of β_0 mismatches due to the mechanism used for tracking connected components. It uses ray-surface intersection to sample a few points over each connected component of the isosurface before extracting it. The number of rays is a user-controlled parameter and its initial position and direction are randomly assigned. DelIso is likely to extract the biggest connected component and, occasionally, it misses small components. It is important to say that the ray-sample based scheme tends to work fine in practical applications where small surfaces are not present. The invariant mismatches for β_1 and β_2 are computed only if no consistency mismatch happens.

For MCFlow, we applied the verification framework systematically during its implementation/development. Obviously, many bugs were uncovered and fixed over the course of its development. Since we are randomizing the piecewise trilinear field, we are likely to cover all possible Marching Cubes entries and also different cube combinations. As verification tests have been applied since the very beginning, all detectable bugs were removed, resulting in no mismatches. The downside of MCFlow, though, is that typical bad quality triangles appearing in Marching Cubes become even worse in MCFlow, because

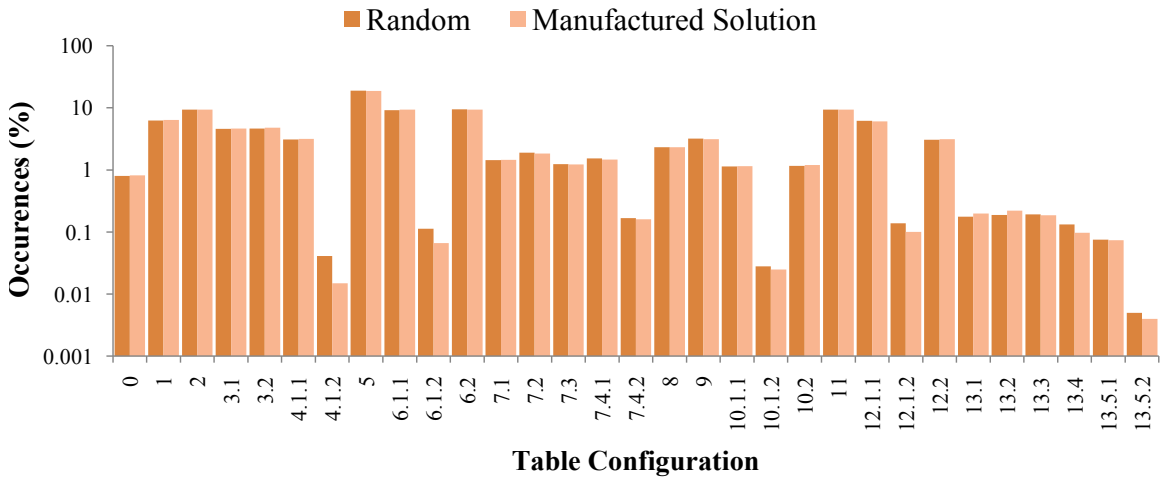


Figure 3.6. The horizontal axis shows the case and subcase numbers for each of the 31 Marching Cubes configurations described by Lopes and Brodlie [94]. The dark bars show the percentage of random fields that fit a particular configuration. The light bars show the percentage of random fields which fit a particular configuration *and* do not violate the assumptions of our manufactured solution. Our manufactured solution hits all possible cube configurations.

cubes of different sizes are glued together. MCFlow geometrical convergence is presented in the supplementary material [147].

3.6 Discussion and Limitations

3.6.1 Quality of manufactured solutions

In any use of MMS, one very important question is that of the quality of the manufactured solutions, since it reflects directly on the quality of the verification process. Using random solutions, for which we compute the necessary invariants, naturally seems to yield good results. However, our random solutions will almost always have nonidentical values. This raises the issue of detecting and handling degenerate inputs, such as the ones arising from quantization. We note that most implementations use techniques such as Simulation of Simplicity [35] (for example, by arbitrarily breaking ties using node ordering) to effectively keep the facade of nondegeneracy. However, we note that developing manufactured solutions specifically to stress degeneracies is desirable when using verification tools during development. We decided against this since different implementations might employ different strategies to handle degeneracies and our goal was to keep the presentation sufficiently uniform.

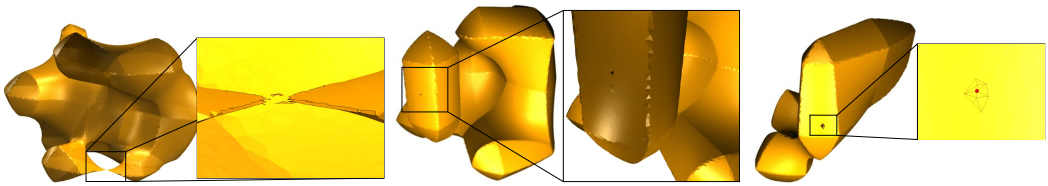


Figure 3.7. DelIso mismatch example. From left to right: holes in C^0 regions; single missing triangle in a smooth region; duplicated vertex (the mesh around the duplicated vertex is shown). These behavior induce topology mismatches between the generated mesh and the expected topology.

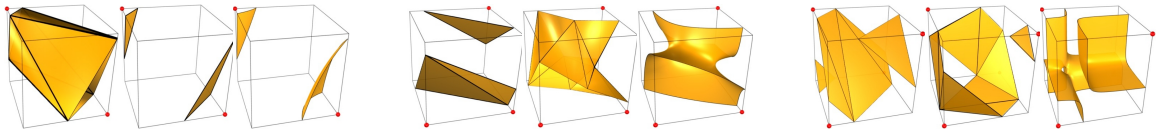
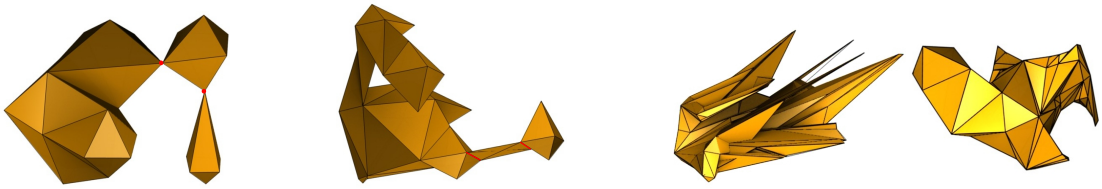


Figure 3.8. MC33 mismatch example. From left to right: problem in the case 4.1.2, 6.1.2, and 13.5.2 of marching cube table (all are ambiguous). Each group of three pictures shows the obtained, expected, and implicit surfaces. Our verification procedure can detect the topological differences between the obtained and expected topologies, even for ambiguous cases.



(a) SnapMC (0.3) (b) Matlab (c) MCFlow

Figure 3.9. Mismatches in topology and geometry. (a) SnapMC generates non-manifold surfaces due to the snap process. (b) Matlab generates some edges (red) that are shared by more than two face. (c) MCFlowbefore (left) and after (right) fixing a bug that causes the code to produce the expected topology, but the wrong geometry.

3.6.2 Topology and Geometry

This work extends the work by Etiene et al. [42] toward including topology in the loop of verification for isosurface techniques. The machinery presented herein combined with the methodology for verifying geometry comprises a solid battery of tests able to stress most of the existing isosurface extraction codes.

To illustrate this, we also submitted MC33 and MCFlow techniques to the geometrical test proposed by Etiene, as these codes have not been geometrically verified. While MC33 has geometrical behavior in agreement with Etiene's approach, the results presented in

Section 3.5 show it does not pass the topological tests. On the other hand, after ensuring that MCFlow was successful regarding topological tests, we submitted it to the geometrical analysis, which revealed problems. Figure 3.9(c) shows an example of an output generated in the early stages of development of MCFlow before (left) and after (right) fixing the bug. The topology matches the expected one (a topological sphere); nevertheless, the geometry does not converge.

3.6.3 SMT vs. DT

The verification approach using digital surfaces generates detailed information about the expected topology because it provides β_0, β_1, and β_2. However, verifying isosurfaces with boundaries would require additional theoretical results, as the theory supporting our verification algorithm is only valid for surfaces without boundary. In contrast, the verification methodology using stratified Morse theory can handle surfaces with boundary. However, SMT only provides information about the Euler characteristic, making it harder to determine when the topological verification process fails. Another issue with SMT is that if a code incorrectly introduces topological features so as to preserve χ, then no failure will be detected. For example, suppose the surface to be reconstructed is a torus, but the code produces a torus plus three triangles, each one sharing two vertices with the other triangles

Table 3.1. Rate of invariant mismatches using the PL manifold property, digital surfaces, and stratified Morse theory for 1000 randomly generated scalar fields (the lower the rate the better). The invariants β_1 and β_2 are computed only if the output mesh is a 2-manifold without boundary. *We run correctness tests in all algorithms for completeness and to test tightness of the theory: algorithms that are not topology-preserving should fail these tests.* The high number of DelIso, SnapMC, and Matlab mismatches are explained in Section 3.5.1. [1] indicates zero snap parameter and [2] indicates snap value of 0.3.

	Consistency (%)	Correctness (%)				
	Disk	Digital Surfaces				SMT
		β_0	β_1	β_2	χ	χ
Afront	0.0	35.9	22.8	35.9	47.5	25.5
Matlab	19.7	32.2	18.9	20.5	49.3	70.3
VTKMC	0.0	27.6	23.2	27.6	43.5	70.7
Macet	0.0	54.3	20.9	54.3	64.0	100.0
SnapMC[1]	0.0	45.0	25.4	45.0	57.3	72.0
SnapMC[2]	53.7	41.6	17.3	23.1	87.1	74.0
MC33	0.0	2.4	1.1	2.4	3.4	5.4
DelIso	19.1	24.4	0.1	20.0	37.2	33.2
MCFlow	0.0	0.0	0.0	0.0	0.0	0.0

but not an edge. In this case, torus plus three "cycling" triangles also has $\chi = 0$, exactly the Euler characteristic of the single torus. In that case, notice that the digital surface-based test would be able to detect the spurious three triangles by comparing β_0. Despite being less sensitive in theory, SMT-based verification revealed similar problems as the digital topology tests have. We believe this effectiveness comes in part from the randomized nature of our tests.

3.6.4 Implementation of SMT and DT

Verification tools should be as simple as possible while still revealing unexpected behavior. The pipeline for geometric convergence is straightforward and thus much less error-prone. This is mostly because Etiene et al.'s approach uses analytical manufactured solutions to provide information about function value, gradients, area, and curvature. In topology, on the other hand, we can manufacture only simple analytical solutions (e.g., a sphere, torus, double-torus, etc.) for which we know topological invariants. There are no guarantees that these solutions will cover all cases of a trilinear interpolant inside a cube. For this reason, we employ a random manufactured solution and must then compute explicitly the topological invariants. A point which naturally arises in verification settings is that the verification code is another program. How do we verify the verifier?

First, note that the implementation of either verifier is simpler than the isosurfacing techniques under scrutiny. This reduces the chances of a bug impacting the original verification. In addition, we can use the same strategy to check if the verification tools are implemented correctly. For SMT, one may compute χ for an isovalue that is greater than any other in the grid. In such case, the verification tool should result in $\chi = 0$. For DT, we can use the fact that Majority Interpolation always produces a 2-manifold. Fortunately, this test reduces to check for two invalid cube configurations as described by Stelldinger et al. [160]. Obviously, there might remain bugs in the verification code. As we have stated before, a mismatch between the expected invariants and the computed ones indicates a problem *somewhere* in the pipeline; our experiments are empirical evidence of the technique's effectiveness in detecting implementation problems.

Another concern is the performance of the verification tools. In our experiments, the invariant computation via SMT and DS is faster than any isosurface extraction presented in this work, for most of the random grids. In some scenarios, DS might experience a slowdown because it refines the grid in order to eliminate ambiguous cubes (the maximum number of refinement is set to 4). Thus, both SMT and DS (after grid refinement) need to perform a constant number of operations for each grid cube to determine the digital surface (DS) or

critical points (SMT). In this particular context, we highlight the recent developments on certifying algorithms, which produce both the output and an *efficiently checkable certificate of correctness*[103].

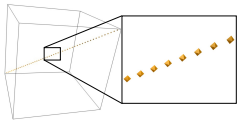
3.6.5 Contour Trees

Contour trees [18] are powerful structures to describe the evolution of level-sets of simply connected domains. It normally assumes a simplicial complex as input, but there are extensions to handle regular grids[126]. Contour trees naturally provide β_0, and they can be extended to report β_1 and β_2. Hence, for any isovalue, we have information about all Betti numbers, even for surfaces with boundaries. This fact renders contour trees a good candidate for verification purposes. In fact, if an implementation is available, we encourage its use so as to increase confidence in the algorithm's behavior. However, the implementation of a contour tree is more complicated than the techniques presented here. For regular-grids, a divide-and-conquer approach can be used along with oracles representing the split and join trees in the deepest level of the recursion, which is non-trivial. Also, implementing the merging of the two trees to obtain the final contour tree is still involving and error-prone. Our approach, on the other hand, is based on regular grid refinement and voxel selection for the DT method and critical point computation and classification for the SMT method. There are other tools, including contour trees, that could be used to assess topology correctness of isosurface extraction algorithms, and an interesting experiment would be to compare the number of mismatches found by each of these tools. Nevertheless, in this work, we have focused on the approaches using SMT and DT because of their simplicity and effectiveness in finding code mistakes in publicly available implementations. We believe that the simpler methodologies we have presented here are more likely to be adopted during development of visualization isosurfacing tools.

3.6.6 Topology of the underlying object

In this work, we are interested in how to effectively verify topological properties of codes which employ trilinear interpolation. In particular, this means that our verification tools will work for implementations other than marching methods (for example, DelIso is based on Delaunay refinement). Nevertheless, in practice, the original scalar field will not be trilinear, and algorithms which assume a trilinearly interpolated scalar field might not provide any topological guarantee regarding the reconstructed object. Consider, for example, a piecewise linear curve γ built by walking through diagonals of adjacent cubes $c_i \in \mathcal{G}$ and define the distance field $d(x) = \min\{||x - x'|| \text{ such that } x' \in \gamma\}$. The isosurface $d(x) =$

α for any $\alpha > 0$ is a single tube around γ. However, none of the implementations tested could successfully reproduce the tubular structure for all $\alpha > 0$. This is not particularly surprising, since the trilinear interpolation from samples of d is quite different from the d.



The inline figure on the right shows a typical output produced by VTK Marching Cubes for the distance field $d = \alpha$. Notice, however, that this is not only an issue of sampling rate because if the tube keeps going through the diagonals of cubic cells, VTK will not be able reproduce $d = \alpha$ yet. Also recall that some structures cannot even be reproduced by trilinear interpolants, as when γ crosses diagonals of two parallel faces of a cubic cell, as described in [21, 126]. The aspects above are not errors in the codes but reflect software design choices that should be clearly expressed to users of those visualization techniques.

3.6.7 Limitations

The theoretical guarantees supporting our manufactured solution rely on the trilinear interpolant. If an interpolant other than trilinear is employed, then new results ensuring homeomorphism (Theorem 4.1) should be derived. The basic infrastructure we have described here, however, should be appropriate as a starting point for the process.

3.7 Conclusion and Future Work

We extended the framework presented by Etiene et al. [42] by including topology into the verification cycle. We used machinery from digital topology and stratified Morse theory to derive two verification tools that are simple and yet capable of finding unexpected behavior and coding mistakes. We argue that researchers and developers should consider adopting verification as an integral part of the investigation and development of scientific visualization techniques. Topological properties are as important as geometric ones, and they deserve the same amount of attention. It is telling that the only algorithm that passed all verification tests proposed here is the one that used the verification procedures *during* its development. We believe this happened because topological properties are particularly subtle and require an unusually large amount of care.

The idea of verification through manufactured solutions is clearly problem-dependent and mathematical tools must be tailored accordingly. Still, we expect the framework to enjoy a similar effectiveness in many areas of scientific visualization, including volume rendering, streamline computation, and mesh simplification. We hope that the results of this work further motivate the visualization community to develop a culture of verification.

Acknowledgments

We thank Thomas Lewiner and Joshua Levine for help with MC33 and DelIso codes respectively. This work was supported in part by grants from NSF (grants IIS-0905385, IIS-0844546, ATM-0835821, CNS-0751152, OCE-0424602, CNS-0514485, IIS-0513692, CNS-0524096, CCF-0401498, OISE-0405402, CCF-0528201, CNS-0551724, CMMI 1053077, IIP 0810023, CCF 0429477), DOE, IBM Faculty Awards and PhD Fellowship, the US ARO under grant W911NF0810517, ExxonMobil, and Fapesp-Brazil (#2008/03349-6).

CHAPTER 4

PRACTICAL CONSIDERATIONS ON THE TOPOLOGICAL CORRECTNESS OF MARCHING CUBES 33

Isosurface extraction techniques can be divided into two classes according to their topological guarantees, namely, consistency or correctness. Topologically *consistent* techniques produce surfaces that are piecewise-linear (PL) manifolds (*i.e.*, crack-free surfaces), except at the boundary of the domain. Topologically *correct* techniques produce a PL-manifold homeomorphic to the surface induced by a given interpolant, such as the trilinear interpolant. Although there are many topologically *consistent* MC-based techniques, only a handful are topologically *correct*. Marching Cubes 33 is one of the first MC-based algorithms that preserves the topology of the trilinear interpolant.

Topological correctness increases the complexity of isosurface extraction algorithms. The many isosurface configurations possible for a given interpolant in a cubic grid makes both the algorithm and its implementation a challenging task. As algorithms and implementations become more complex, issues may be overlooked and remain hidden in the multitude of (pseudo-) lines of code. Throughout years of research, it has been shown that some topologically correct techniques, including MC33, have issues that prevent correctness [41, 94, 120]. In particular, the work presented in the Chapter 3 [41] shows that the MC33 implementation by Lewiner *et al.* [89, 90], fails to produce topologically correct isosurfaces. Nevertheless, no explanation is given for the problem source, nor possible solutions. As we continued our study of the MC33 implementation, we realized that the source of the problem was not merely implementation bugs but the core ideas behind the implemented algorithm. In this chapter, we address issues with Chernyaev's original algorithm, its extension, and its implementation. This work is another step towards closing an existing gap in the topological correctness of Marching Cubes 33.

The subtleties involved in the correctness of isosurface extraction techniques are sometimes difficult to grasp in the ordinary paper medium. Both the geometry and topology

inside grid voxels are often complex and challenging to understand, study and replicate (*e.g.*, see Figures 9 and 10 in [121]). As an attempt to bridge this gap, we build on recent efforts towards *executable papers* [78, 167]. Executable papers extend the traditional paper/digital counterpart by including tools that allow readers to interact, explore and verify experiments more easily. In this work, we use executable papers to increase the reproducibility of our results.

In this chapter, our contributions are the following:

- We explain and address three algorithmic issues and one non-trivial implementation issue with Marching Cubes 33. In particular, we solve an issue with the core MC33 disambiguation procedure that, as far as we know, has not been addressed elsewhere. Hence, we close an existing gap in the MC33 literature.
- We make our results reproducible. CrowdLabs [167] and Vistrails [45] are used to create an executable paper that can reproduce the results shown in the following sections.
- We provide datasets that can be used to verify the correctness of any topologically correct isosurface extraction technique.

A by-product of this work is a thorough analysis of both the MC33 algorithm and its implementation that can be used by anyone interested in the use or development of correct isosurface extraction algorithm based on MC33. The results of our efforts are materialized into an extended version of the MC33 implementation [89], henceforth called Corrected-MC33 (C-MC33).

4.1 Related Work

Soon after the publication of the MC algorithm, the quest for a topologically correct isosurface extraction technique began. A number of approaches were proposed for dealing with cracks, face ambiguity, and, lastly, interior ambiguity. Dürst [33] was the first to point out that some MC cases allow multiple triangulation. A consequence of this is that MC does not always generate topologically consistent surfaces. This problem arises due to the *ambiguity problem*; and the Asymptotic Decider [122] provides a simple and elegant solution to face ambiguity.

The ambiguity problem also occurs in the interior of a voxel. Natarajan [118] was the first to address this problem by adding four new cases to the standard MC triangulation table (subcases of cases 3, 4, 6, and 7). To find the correct subcases, the author proposed

a disambiguation procedure based on both face and interior critical points. Nevertheless, the method misses the possibility of two interior critical points in case 7, consequently the proposed algorithm may generate a surface with the incorrect topology [15, 120].

Using a different approach, Chernyaev [21] extended the original MC table to 33 cases – hence MC33; this extension included all the subcases for each ambiguous case. This author used the Asymptotic Decider and a new interior ambiguity test to discriminate among subcases. Lewiner *et al.* [90] provided a practical open-source implementation of the Chernyaev algorithm. Matveyev [101], introduced an isosurface technique that is also based on an extended table and used the intersections of isosurfaces with cube diagonals to determine the correct case.

Lopes and Brodlie [94] extended the tests proposed by Natarajan. The goals of the work are threefold: i) extract topologically correct isosurfaces; ii) produce geometrically accurate isosurface; iii) allow continuity with respect to changes in threshold and data. Nevertheless, as in Natarajan's work, the method missed the possibility of two interior critical points in case 7 [94]. Cignoni *et al.* [23] also used the test proposed by Natarajan to reconstruct topologically correct isosurfaces. The work of Theisel [163] uses Bézier patches to build G^1 continuous isosurfaces that are topologically correct. Nielson [121] lists all possible cases of a trilinear interpolant inside a cubic grid and builds a topologically correct MC using a three stage algorithm for surface polygonization.

The past two decades have also produced a number of isosurface techniques that are not MC-based. Dual Contouring [69] (DC) is a robust, crack-free, isosurface extraction technique that works on the dual grid. Several improvements over Dual Contouring have been proposed: Schaefer *et al.* [145] address the issue of non-manifold surfaces generated by DC; Varadhan *et al.* [177] combine a signed distance field with DC to reconstruct details such as thin features; and Zhang *et al.* [188] use DC for topology-preserving simplification of isosurfaces. Note that none of these techniques are intended to preserve the topology of the trilinear interpolant. Dey and Levine [30] presented an algorithm that computes a Delaunay triangulation based on the intersection between the isosurface and the 3D Voronoi diagram. Another paradigm for isosurface extraction is the advancing front method. Advancing front algorithms build a triangulated surface by progressively adding triangles to an implicit surface [55], possibly creating several fronts that are simultaneously advanced one triangle at a time. A number of extensions have been proposed for advancing front techniques [149, 150, 155].

In the following sections, we focus on MC33. Note that, although many of the algorithms

presented previously are topologically consistent, only a handful of them are topologically correct [21, 30]. Also, the implementation of a topologically correct isosurface extraction algorithm is non-trivial. Hence, once the algorithm is implemented, topological guarantees, both consistency and correctness, may be lost because of algorithm or implementation issues, as shown in the work of Etiene *et al.* [41]. Although it has been ten years since the publication of MC33, we believe it is important to correct a mistake in the algorithm that has gone unnoticed since Chernyaev published it almost 20 years ago. In this work we aim to close an existing gap in the MC33 literature. Furthermore, we aim not only to provide a correct algorithm but verify that our modified implementation is faithful to the correct algorithm. We explain the issues and propose solutions for both algorithm and implementation. We note that "MC33" may refer to either the Marching Cubes 33 *algorithm* presented in Chernyaev's work [21] or its *implementation*, as in Lewiner *et al.* [90] depending on the context.

4.2 Preliminaries

In this section we present the notation that will be used throughout the paper. We also briefly review the main concepts behind Chernyaev's algorithm and Lewiner *et al.*'s improvements to it. Let G be a rectilinear grid with scalar values associated with each vertex $x_j \in G$. Let $g : \mathbb{R}^3 \to \mathbb{R}$ be a piecewise-trilinear interpolation function defined on G. Given an isovalue λ, the isosurface S_λ is defined as the set of points for which $g(x) = \lambda$. For each voxel $v_i \subset G$, and $x \in v_i$, $g(x) = g_i(x)$ where g_i is the trilinear interpolant inside the cubic cell v_i.

The output of MC-based algorithms is a piecewise-linear mesh M_λ, and we say that an algorithm and its implementation are *topologically correct* if M_λ is homeomorphic to S_λ. Without loss of generality, we assume that $\lambda = 0$, and thus $S_\lambda = S_0 = S$. We say that a point x is *positive* (*negative*) if $g(x) > 0$ ($g(x) < 0$).

Given a voxel v_i, and a cutting-plane P parallel to one of v_i's faces, define $f_i : \mathbb{R}^2 \to \mathbb{R}$ as the bilinear interpolant along P. Note that $f_i(x) = g_i(x)$ for $x \in P$. Throughout the text, we deal with a single voxel v; thus, we omit the subscript i. We also assume that v and P are defined in the domains $[0, 1]^3$ and $[0, 1]^2$, respectively.

4.2.1 Chernyaev's MC33

The two pillars of Marching Cubes 33's topological correctness are Nielson and Hamann's Asymptotic Decider and Chernyaev's interior ambiguities test; together these solve the face ambiguity and interior ambiguity problems in the Marching Cubes 33 algorithm. A face

ambiguity occurs when face vertices have alternating signs. That is, one face diagonal is positive (both vertices are positive) and the other is negative (both vertices are negative). In this case, the signs of the face vertices are insufficient to determine the correct way to triangulate the isosurface (see Figure 4.1). Similarly, an interior ambiguity occurs when the signs of the cube vertices are insufficient to determine the correct surface triangulation, *i.e.*, when multiple triangulations are possible for the same cube configuration.

The idea behind the Asymptotic Decider is to verify the face saddle sign and compare it to the sign on the face vertices. A positive saddle means that the positive face vertices are connected; consequently, the positive face vertices are separated if the face saddle point is negative (see Figure 4.1). To compute the face saddle sign, the saddle point position x_c must be computed [21]:

$$x_c = \left(\frac{A-D}{A+C-B-D}, \frac{A-B}{A+C-B-D} \right), \tag{4.1}$$

where A, B, C and D are the scalar values at the face vertices (see Figure 4.1). The sign of x_c can easily be checked by replacing Equation (4.1) into the bilinear interpolant:

$$f(x_c) = \frac{AC-BD}{A+C-B-D}. \tag{4.2}$$

For an ambiguous face, assuming A, C positive and B and D negative, the denominator of the Equation (4.2) is always positive (see Figure 4.1). Then, the face ambiguity is solved by evaluating the sign of $f(x_c)$ numerator.

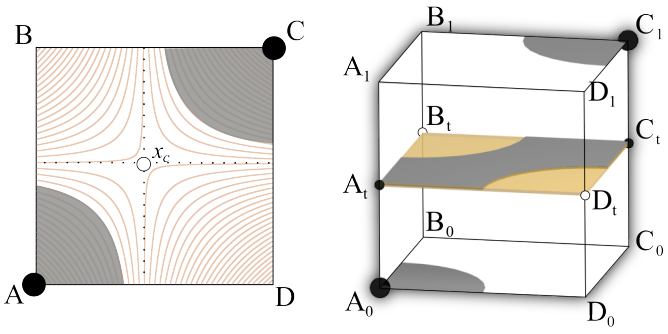


Figure 4.1. Asymptotic Decider (left) and MC33 interior ambiguity test for MC case 4 (right). The gray areas represent regions with positive scalar values, and the capital letters represent the scalar value at each vertex. In the left image, we observe that $f(x_c) < 0$, where x_c is the saddle point position. Positive areas will be connected if $f(x_c) > 0$. The orange squared plane shown in the right image represents the cutting-plane. The goal of the MC33 algorithm is to find a cutting-plane such that the gray areas in the top and bottom planes are joined in the interior.

Due to the interior ambiguity, the Asymptotic Decider alone cannot solve the topological correctness problem. Chernyaev uses the idea behind the Asymptotic Decider to solve the interior ambiguity problem. The proposed test uses a sweeping cutting-plane to evaluate the behavior of the trilinear interpolant inside the cube.

Given a cube with a ambiguous configuration, define the scalar values at the base and top planes as $A_0B_0C_0D_0$ and $A_1B_1C_1D_1$, respectively. Let A_0 and C_1, the vertices to be tested, be positive. Observe that, although A_0 and C_1 belong to opposite cube faces, they can be connected through the cube interior. In other words, there may exist a path from A_0 to C_1 passing through the voxel interior for which all points belonging to that path are positive. To determine whether A_0 and C_1 are connected, Chernyaev begins by observing that the saddle points at the top and base cube faces are negative, *i.e.*, Equation (4.2) is negative at the bottom and top faces. It follows that the denominator of $f(x_c)$ is positive because:

$$A_0C_0 - B_0D_0 < 0 \tag{4.3}$$

$$A_1C_1 - B_1D_1 < 0. \tag{4.4}$$

Then, if there is a plane cutting the cube such that its saddle point is positive, it means that there is a positive area crossing the cube, *i.e.* the positive vertices are connected inside the cube. In other words, the Chernyaev interior test searches for a t for which:

$$F(t) = A_tC_t - B_tD_t > 0. \tag{4.5}$$

This can be achieved by solving a second order equation in t. Replacing $X_t = X_0 + (X_1 - X_0)t$, $X \in \{A, B, C, D\}$ and $t \in [0, 1]$ in Equation (4.5) one obtains a second order equation in t:

$$F(t) = A_tC_t - B_tD_t \tag{4.6}$$

$$= at^2 + bt + c, \tag{4.7}$$

where a, b, and c are functions of A, B, C, and D (see A). Chernyaev concludes that positive vertices A_0 and C_1 are connected through the cube interior if:

1. $a < 0$;
2. $t_{\max} = -b/2a \in (0, 1)$;
3. $F(t_{\max}) > 0$.

If one of the above conditions fails, the positive vertices are separated.

4.2.2 Lewiner *et al.*'s MC33

Lewiner *et al.* [90] proposed a modification of Chernyaev's interior test. In this modification, they use an alternative method for computing the height plane t for most ambiguous cases. For cases 6, 7, 12, and 13, the authors compute the height t based on the barycenter of the end vertices of an edge e (a cube edge intersected by the isosurface) weighted by the values of the scalar field on these vertices (see [90] for details). In practice, the implementation uses:

$$t_{\text{alt}} = \frac{V_0}{V_0 - V_1}, \tag{4.8}$$

where V_0 and V_1 are the scalar values at the vertices of e. Note that this is equivalent to finding the intersection point between the isosurface S and e. The authors keep the structure of the test proposed by Chernyaev, but condition (i) is not used, and condition (ii) is always true because e is an edge intersected by the isosurface; consequently, $t_{alt} \in (0, 1)$.

Section 4.4 explains why the algorithm proposed by Chernyaev and its modified version proposed by Lewiner *et al.* may fail to extract surfaces that are topologically correct. In the following section, we present the tools we use to detect, debug, and reproduce the issues found in the MC33 algorithm and its implementation.

4.3 Experiments setup

We begin by investigating the source of topological problems in the MC33 implementation [41]. The topological issues described were obtained by systematically stress-testing the MC33 implementation over many topological configurations using the verification framework proposed in Etiene *et al.* [41]. These authors' algorithm can be summarized as follows. (I) A random scalar field G is built by uniformly sampling scalar values in the range $[-1, 1]$ for each $x_j \in G$. (II) The *expected topological invariants* are obtained directly from S, *i.e.*, without extracting the isosurface of interest. The topological invariants used are the Euler characteristic $\chi(S)$ and the Betti numbers $\beta_k(S)$. (III) The MC33 implementation is used to extract a piecewise linear mesh M, and its invariants $\chi(M)$ and $\beta_k(M)$ are computed. (IV) Lastly, the pairs of topological invariants $\{\chi(S), \chi(M)\}$ and $\{\beta_k(S), \beta_k(M)\}$ are compared. A mismatch indicates that a problem has occurred. Nevertheless, as the authors note, a match between invariants does not imply a bug-free code [41]. The verification process does not prove the absence of bugs but only increases one's confidence in its correctness. In this paper, we exploit the fact that when the expected and obtained surfaces are not homeomorphic, a counterexample is given in the form of a scalar field G and a mesh M. We use this information to find and correct errors in MC33.

4.3.1 Reproducibility

As investigators in a mature field within the scientific visualization community, isosurface extraction researchers have developed ways to help other researchers and practitioners reproduce their results. Published journal articles offer a first approximation of reproducibility. Nevertheless, many details regarding implementation, source code, input data, and other types of information are often omitted. Many, but not all, published techniques make source code and input data freely available, and some are part of widely used visualization packages such as VTK [151]. This practice greatly increases the degree of reproducibility of the work. In this paper, we strive to increase the degree of reproducibility of the work presented by making the results shown in Figures 4.6 and 4.7 and in Algorithm 7 fully reproducible. We use CrowdLabs [167] and Vistrails [45, 154] as a platform to achieve this goal. To explore some of the results shown in this paper, the reader may click on individual figure captions and interact with the results via web browser. Figure 4.2 shows the pipeline of an executable paper. We have selected cases in which MC33 fails have provided the respective correct results. In addition, to allow the reader to explore and study the results presented here, he or she can also download the scalar fields and respective topological invariants χ and β used for stress testing MC33. We also provide 10000 Marching Cubes cases grids and randomly generated 5x5x5 grids. This dataset can be used to test any topologically correct isosurface extraction technique.

4.4 Issues with the MC33 algorithm and its implementation

In this section, discuss specific the issues regarding both Chernyaev [21] and Lewiner *et al.*'s [90] work. Because Lewiner *et al.* extends Chernyaev's work, the issues presented in the latter are also part of the former. Specifically, we detail three *algorithmic* issues – two in Chernyaev's MC33 and one in Lewiner *et al.* – and one *implementation* issue. The solutions for the issues raised here will be presented in the next section.

This section is organized as follows. First, we explain an algorithmic problem with the MC33 core disambiguation procedure. This issue has not been discussed in the literature to date. We then discuss a second algorithmic problem related to the triangulation table and the extraction of non-manifold meshes. Although this problem has been discussed in the literature, we discuss it here for completeness and because we provide a simple solution to the problem (see Section 4.5). Next, we show a third algorithmic problem related to the alternative approach proposed by Lewiner *et al.* for computing the height plane t. Lastly, we show a non-trivial problem with the open-source implementation of the MC33.

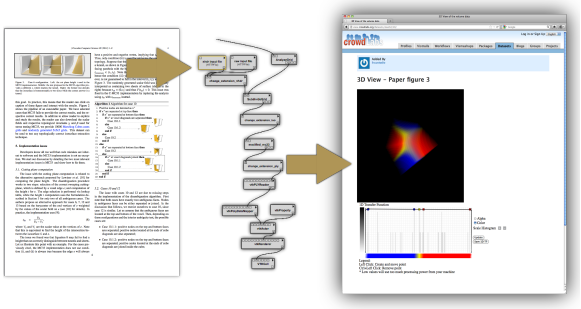


Figure 4.2. The executable paper pipeline. An image is made executable (left), meaning that the reader can launch a request to execute a pipeline in a remote server (middle) and interact with the result in a web browser (right).

4.4.1 Issue I – Case 13.5

Here, we show a problem with the core disambiguation procedure described in Chernyaev's work. To our knowledge, this problem has not been exposed or addressed in the literature.

Case 13 is certainly the most complex table case; all faces are ambiguous, and six subcases are possible. Four of the subcases can be discriminated by using Asymptotic Decider. The remaining cases 13.5.1 and 13.5.2 require Chernyaev's MC33 interior ambiguity resolution method. Recall that the MC33 approach discriminates between tunnels and isolated sheets by finding a cutting-plane for which positive nodes in the cube diagonal are joined by points in the interior of the cubic cell. Cases 13.5.1 and 13.5.2 differ precisely because the positive nodes in case 13.5.2 are connected to one another by the interior points, which is not true for 13.5.1 (see Figure 4.3).

Although it seems that the MC33 methodology described in Section 4.2 fits naturally in this scenario, as it turns out this disambiguation procedure *cannot* be applied for 13.5. Let us illustrate this point with an example. Figure 4.4 shows the expected changes in the sign of the saddle point x_c as a function of the height t. Mathematically

$$x_c(t) \quad = \quad \left(\frac{A_t - D_t}{A_t + C_t - B_t - D_t}, \frac{A_t - B_t}{A_t + C_t - B_t - D_t} \right). \tag{4.9}$$

It follows that the face saddle value (and thus sign) is also defined as a function of t:

$$f(x_c(t)) \quad = \quad \frac{A_t C_t - B_t D_t}{A_t + C_t - B_t - D_t} \tag{4.10}$$

$$= \quad \frac{at^2 + bt + c}{A_t + C_t - B_t - D_t}. \tag{4.11}$$

As can be seen in Figure 4.4, from left to right, as the plane height t changes, the value of the face saddle $f(x_c(t))$ changes from negative to positive to negative and to positive again.

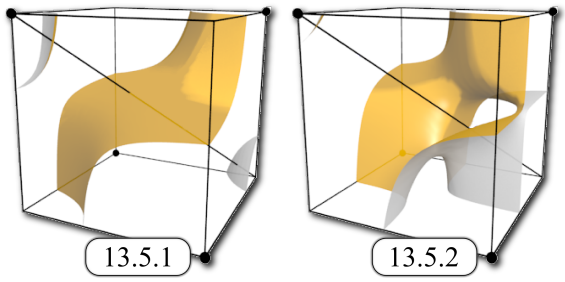


Figure 4.3. Challenging cases for Chernyave's interior test: voxel diagonal has vertices with alternating signs. Case 13.5.2 needs to be oriented correctly. One of the diagonal vertices is isolated from all other vertices in the cube, while the other is faced by the tunnel. In order to determine which vertex is isolated, we apply the same tool used for disambiguation of case 13.5. For case 13.5.1, the orientation of the isosurface have no influence on the topology.

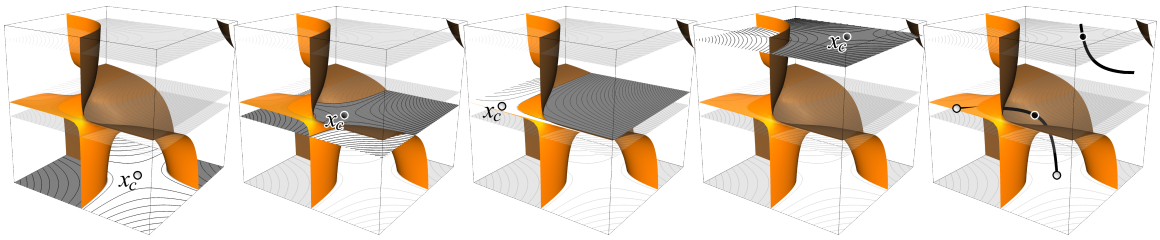


Figure 4.4. Sign changes of the cutting-plane saddle point as a function of the height t. The gray area depicts $f(x) > 0$. The black (resp. white) dots are face saddles with $f(x_c) > 0$ (resp. $f(x_c) < 0$). From left to right, the four leftmost images show the sign of the face saddle points changing from negative to positive to negative and to positive again, respectively. The rightmost image shows the hyperbolic trajectory of the face saddle position $x_c(t)$. The MC33 algorithm fails to track the saddle point sign because it ignores the influence of the hyperbolic trajectory shown here.

These changes occur at the roots t_1 and t_2 of $f(x_c(t))$ and the asymptote of $f(x_c(t))$, *i.e.*, the root t_a of the denominator of f (see left image in Figure 4.5). Thus, in total, three sign changes will occur. The rightmost image in Figure 4.4 shows the path traced by the face saddles $x_c(t)$; as t grows, there is a "jump" not only in the sign of $f(x_c(t))$ but also in the position of $x_c(t)$. The change occurs precisely when the height t passes through the asymptote of $f(x_c(t))$.

Nevertheless, contrary to what is expected, the polynomial $F(t)$ (Equation (4.7)), used by Chernyaev's MC33 algorithm for tracking the sign of the saddle point, is a second order equation on t and thus can only allow for two sign changes. Therefore, the sign tracked by the MC33 algorithm will not match the expected one at some point. Because the sign of the saddle points is embedded in all three conditions for verifying the presence or absence

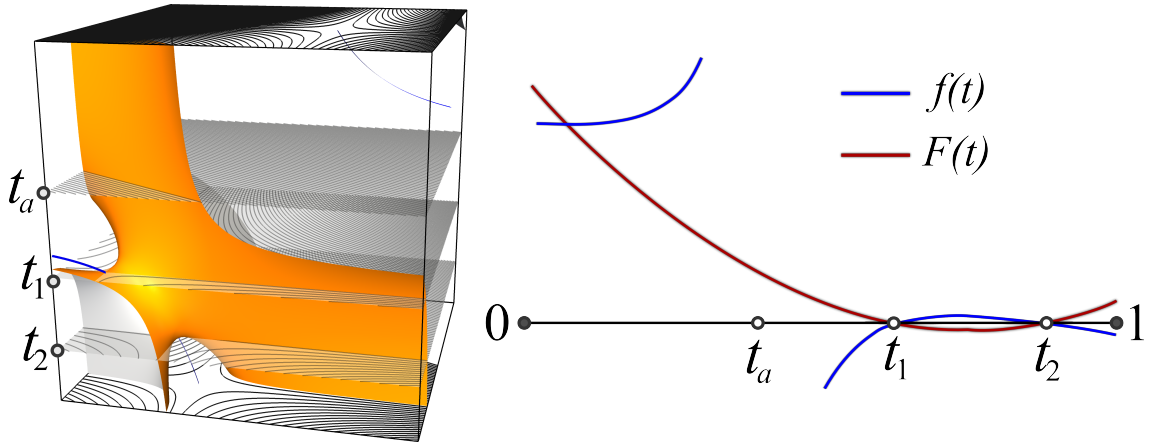


Figure 4.5. Counterexample to Chernyaev's core disambiguation algorithm. The MC33 algorithm incorrectly interprets case 13.5.2 as 13.5.1. The left image shows the zero-level set for case 13.5.2 and cutting-planes at heights t_1, t_2, and t_a, which correspond to both roots of $F(t)$ and the asymptote of $f(x_c(t))$, respectively. The blue ribbon shows the path of the face saddle $x_c(t)$. The right image shows the changes in $f(x_c(t))$ and $F(t)$. According to the three criteria of the MC33 algorithm described in Section 4.2, the upward-facing red parabola defines the absence of a tunnel (condition (i)), which is incorrect. The blue curve, on the other hand, shows the correct sign change.

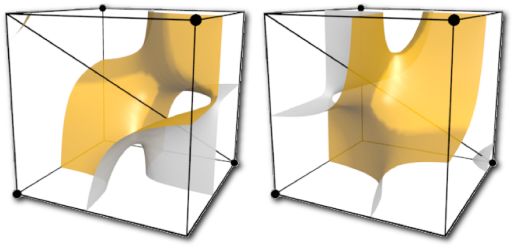
of tunnels, MC33 will eventually provide a wrong result.

The source of the problem can be tracked to Equations 4.3 and 4.4 and the assumption that the denominator of $f(x_c)$ (Equation (4.2)) is positive. These assumptions can easily be verified to be true for case 4, shown in Figure 4.1. However, for case 13, the saddle sign at the top and bottom planes have opposite signs, which contradicts Equations (4.3) and (4.4). In addition, the denominator $A + C - B - D$ of $f(x_c)$ changes its sign at the asymptote of $f(x_c)$, contrary to the assumption that it is always positive. The consequence of incorrectly tracking sign changes is that the three rules used for resolving internal ambiguity will fail for some scalar fields. As an example, Figure 4.5 shows a case 13.5.2 that will mistakenly be taken as case 13.5.1 because $a > 0$ characterizes multiples surface sheets instead of a tunnel (see also A). The problem is not only related to the misclassification of case 13.5.2 as 13.5.1. We have also devised examples in which case 13.5.1 is mistakenly taken as case 13.5.2 because the three criteria shown in Section 4.2 hold. Thus, Chernyaev's interior ambiguity test does not always yield topologically correct isosurfaces.

4.4.1.1 Tunnel orientation

A second minor issue regarding case 13.5.2 is the tunnel orientation of configuration 13.5.2. Once case 13.5.2 is determined, one needs to properly orient the tunnel inside the

voxel. The inline figures show the two possibilities. Both vertices at the voxel diagonal are separated from all other voxel vertices at the voxel faces (note that this is not the case for other vertices). Nevertheless, either the positive or the negative vertex of the cube diagonal will connect with vertices with the same sign through the voxel's interior. This will determine which vertex is isolated and which is facing the tunnel. This problem with the tunnel orientation is not dealt with or mentioned in either Chernyaev or Lewiner *et al.*'s work. Nevertheless, it was briefly mentioned in Etiene *et al.* [41], but no solution to the problem was provided. As the authors observed, the isosurface topology changes if the tunnel orientation is incorrect; thus, it must be oriented correctly. The following section provides a solution for this issue.



4.4.2 Issue II – Non-manifold surfaces

The second algorithmic issue is related to the triangulation table used to build triangulated surfaces. The choice of the correct MC configuration is only part of the process of building an algorithm that preserves the topology of the piecewise-trilinear field. The voxel triangulation table is, in fact, the determinant of the final mesh topology. Chernyaev's original triangulation table contains cases that lead to topologically inconsistent non-manifold meshes in scenarios such as the one shown in Figure 4.6. This problem occurs because the MC33 triangulation table allows faces that are coplanar with the grid voxel faces. Hence, when neighbor voxels have "tunnels" in their interiors, and share an ambiguous, coplanar face, the end result will be non-manifold edges, as shown in Figure 4.6. Because this is an issue with the triangulation table, any topologically correct algorithm whose table is based on Chernyaev's triangulation table will build non-manifold surfaces whether or not the algorithm can correctly distinguish the voxel cases.

This problem was pointed out by Gelder and Wilhelms [48] and is one of the motivations of Lopes and Brodlie's work on topologically correct and geometrically accurate isosurface extraction algorithm [94]. The authors aimed at improving the geometry quality of the trilinear surface patches and consequently solving the topology problem. They achieve this goal by adding points to the voxel faces as well as to the voxel interior. These extra points are placed on the trilinear patch which increases geometry accuracy. They are classified into three different classes and used to extending the contour of the trilinear patch with the

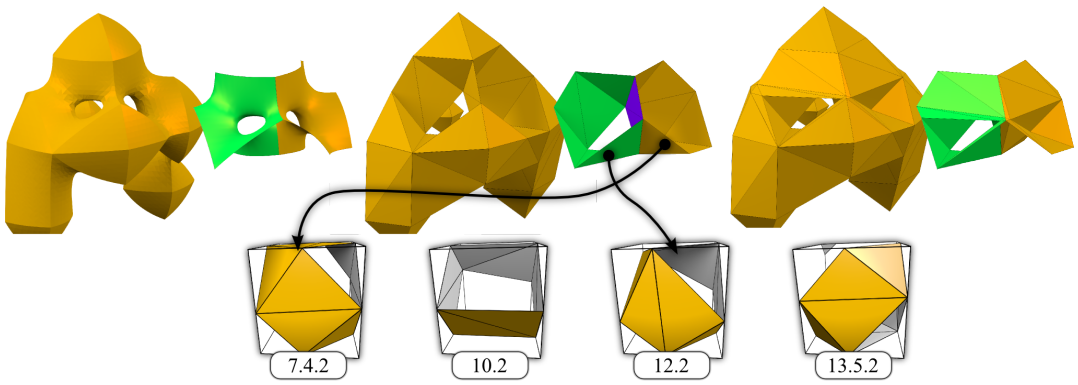


Figure 4.6. Top: Problem with Chernyaev's triangulation table. The figure shows the zero level-set of a $5 \times 5 \times 5$ randomly generated piecewise-trilinear scalar field G (left) and two meshes extracted using the MC33 (center) and C-MC33 (right) algorithms. The isolated voxel patches, shown in green and yellow, represent the two voxels at the center of G. The face shared by two consecutive tunnels, shown in purple, generates non-manifold edges. After one subdivision at the critical point of this case, the problem no longer occurs, and a valid manifold surface is obtained (right). Bottom: Triangulation for tunnels used by Lewiner *et al.* [90]. Each has a face that is coplanar to the voxel faces, which may lead to non-manifold surfaces.

voxel faces. The implementation of this technique becomes intricate and error-prone due to the additional steps required for voxel triangulation.

4.4.3 Issue III – Cutting-plane computation

The third algorithmic issue is related to an MC33 improvement proposed by Lewiner *et al.* [90] for computing the plane height. The problem is that Equation (4.8) may fail to find an appropriate height that can correctly distinguish between tunnels and surface sheets. Let us illustrate this point with an example. For the cases previously cited, two of the conditions in the Chernyaev interior test described in Section 4.2 are not used. The MC33 implementation does not use condition (i), and (ii) is always true because the edge e will always have a positive and a negative vertex, implying that $t_{\text{alt}} \in (0, 1)$. Thus, only condition (iii) is used in retrieving the correct voxel topology. Suppose that the scalar field in a given voxel defines a tunnel, as shown in the left image in Figure 4.7. In this case, to retrieve the correct topology, $F(t)$ should be a downward-facing parabola with both roots $t_1, t_2 \in (0, 1)$, $t_1 < t_2$, and $t_{\max} \in (t_1, t_2)$. In this case, $F(t) > 0$ only for $t \in (t_1, t_2)$; hence, $F(t_{\max}) > 0$, and a tunnel is retrieved according to condition (iii). The problem with the alternative approach is that, as shown in Figure 4.7, Equation (4.8) is not guaranteed to fall within the (t_1, t_2) interval, which implies that the scalar field may be incorrectly interpreted as containing two sheets of surface (shown on the right). In other words, because $t_{\text{alt}} \in (0, t_1)$

and $F(t_{\text{alt}}) < 0$, condition (iii) verifies the absence of a tunnel.

4.4.4 Issue IV – Case 10

The last issue described in this work is related to the implementation of MC33. Developers know all too well that code mistakes are inherent to software and the MC33 implementation is not an exception.

Due to a missing step in the implementation of the disambiguation algorithm, MC33 fails to correctly resolve the ambiguity in cases 10 and 12. Note that both cases have exactly two ambiguous faces and the nodes in ambiguous faces can be either separated or joined. In the discussion that follows, we restrict ourselves to case 10; case 12 is similar.

Let us assume that the ambiguous faces are located at the top and bottom of the voxel. Then, depending on the sign of the face saddles and the interior ambiguity test, one can identify the correct case. Chernyaev proposes the following [21]:

- Case 10.1.1: the positive nodes on both faces are separated, and the positive nodes at cube diagonals are also separated;
- Case 10.1.2: the positive nodes on both faces are separated, and the positive nodes at the cube diagonals are not;
- Case 10.2: the positive nodes are connected on the top and separated on the bottom face.

The cases shown above assume that the positives nodes at the top face are separated. But a similar reasoning must be applied to cases in which the positives nodes at the top faces are joined. In the Lewiner *et al.*'s implementation the possibility that the positive nodes at the top faces are joined is missing.

4.5 Solutions

We present solutions for the four issues raised in the previous section.

4.5.1 Issue I – Case 13.5

The disambiguation of case 13.5 has been approached in different ways for different frameworks for isosurface extraction. For example, Nielson [121] presents an algorithm that is concerned with connectivity along edges, faces and the voxel interior. The author presents a detailed description of the behavior of the trilinear interpolant inside the cubic grid and uses these descriptions to solve the ambiguity problem in the interior. Lopes and Brodlie

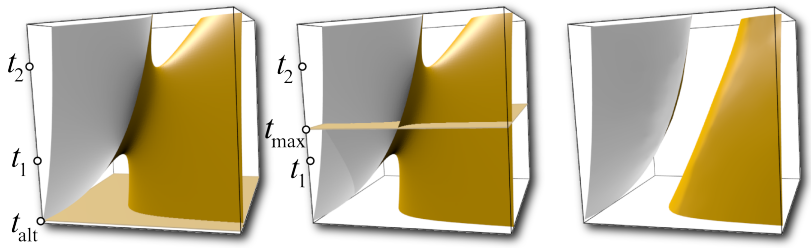


Figure 4.7. Case 6 configuration. Left: the cut plane height $t = t_{alt} > 0$ used in the MC33 *implementation*. Middle: the test proposed in the MC33 *algorithm* provides a different $t = t_{max} > 0$, which reaches the tunnel. Right: the former test decides that the isosurface is homeomorphic to two discs whereas the correct answer is a tunnel.

[94], on the other hand, use critical points in order to resolve some ambiguities. In this case, the sign of the critical point determines the correct configuration. Unfortunately, the above solutions do not seamlessly integrate with the MC33 algorithm. The core idea for solving interior ambiguity, namely, that tunnels can be detected by a sweeping plane through the voxel, is absent in both approaches. This motivated us to devise a simpler solution that follows the idea presented in the original algorithm.

We solve this problem by proposing a new interior test that uses the fact that case 13.5.2 requires both *roots* t_1 and t_2 of $f(x_c(t))$ and the associated saddle points to be inside the voxel. First, recall that $x_c(t)$ tracks the path of the face saddle inside the voxel as a function of height plane at height t, and $f(x_c(t))$ tracks the value (and thus the sign) of that saddle. Both functions are illustrated in the rightmost image in Figure 4.4, in which the black hyperbolic curves represent the path of $x_c(t)$ and the color of the circles represents the sign of the face saddle at a given point (white and black circles are points with negative and positive values, respectively). For case 13.5.2, the path traced by the curve $x_c(t)$ must intersect the isosurface tunnel twice, once at each of the roots t_1 and t_2 of $f(x_c(t))$. This implies that both saddle points $x_c(t_1)$ and $x_c(t_2)$ must lie inside the voxel. This is not the case for 13.5.1 because the face saddle can cross the middle sheet at most once. Therefore, it suffices to verify that both roots of $f(x_c(t))$ and its saddle points are inside the voxel. Algorithm 6 illustrate our solution. Our algorithm is very simple, and does not require the computation of the critical points of the trilinear interpolant, or a detailed description of its behavior inside a voxel. Our algorithm uses the ideas proposed by Chernyaev in order to fix an algorithmic problem in his work. We have implemented and tested this solution on C-MC33 using over 10000 randomly generated cases 13.5.

Algorithm 6 A simple disambiguation procedure for Case 13.5

CASE 13.5(a, b, c)
▷ Let t_1 and t_2 be the roots of $at^2 + bt + c$ (Equation (4.7))
1 **if** $t_1, t_2 \in (0, 1)$ and $x_c(t_1), x_c(t_2) \in (0, 1)^2$
2 **then return** Case 13.5.2
3 **else return** Case 13.5.1

4.5.1.1 Tunnel orientation

To find the correct tunnel orientation one can use the sign of any point between the roots t_1 and t_2. This is because any point in this range must have the same sign as the critical points of the trilinear interpolant for case 13.5.2. This can be seen in the black path shown in the rightmost image in Figure 4.4 and from the graph in Figure 4.5. All points between roots t_1 and t_2 will have the same sign, which is the sign of the "interior" of the tunnel. Thus, we compare the sign of $f((t_1 + t_2)/2)$ with the sign of both vertices of the voxel diagonal. The tunnel will face the vertex with the same sign as $f((t_1 + t_2)/2)$, whereas the other vertex with be isolated from all cube vertices. Figure 4.8 illustrate this scenario. Note that Lopes and Brodlie [94] used the sign of the critical points of the trilinear interpolant to retrieve the correct tunnel orientation. We provide a simpler solution that fits nicely with Chernyaev's framework.

4.5.2 Issue II – Non-manifold surfaces

A possible solution to this problem involves to post-processing the mesh to remove non-manifold features. Although many works in the literature proposed methods for fixing meshes (see [68] for an excellent survey), these are mainly focused on retrieving a valid manifold mesh. Topologically correct algorithms, on the other hand, require that the topology of the trilinear interpolant be preserved. In addition, mesh repairing techniques may mask implementation issues by fixing them, which complicates the verification process.

We use an alternative approach that does not require any changes in the MC33 triangulation table. An interesting fact is that this problem has a low probability of being generated at random and an even lower probability of occurring in real-world datasets. For example, as shown in Figure 4.11 for the Skull dataset this problem appeared six times in total for 50 distinct isosurfaces. In our tests, it occurred only once in 10000 randomly generated $5 \times 5 \times 5$ scalar fields. Thus, instead of implementing the approach of Lopes and Brodlie's, we adopt a simpler solution that takes advantage of the fact that this is a rare event. Non-manifold sur-

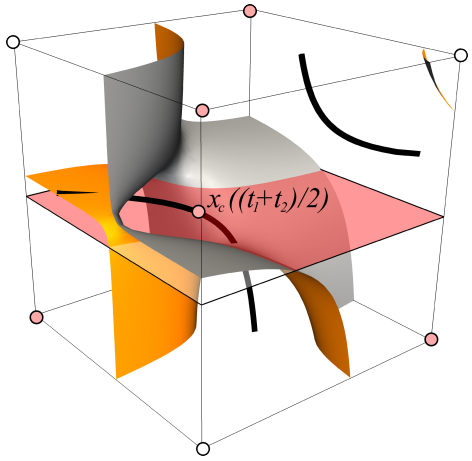
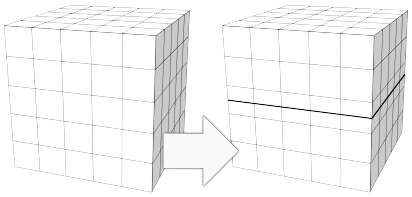


Figure 4.8. Solution to the orientation problem. The red (resp. white) dots represent regions with positive (resp. negative) scalar values.The cutting-plane location is at $(t_1+t_2)/2$. The sign of $f(x_c((t_1+t_2)/2))$ determines the tunnel orientation.

faces are created when two adjacent voxels that share an ambiguous face have tunnels in the voxel interior. By splitting both voxels at the critical point of that face, the face ambiguity is eliminated [15]. To simplify the algorithm, we split not only the voxels sharing the ambiguous face but all faces in the volume slice that contains that face (see inset). Assuming an input



of size $n \times n \times n$, the increase in the number of voxels will be $k(O(n^2))$, where k is the number of subdivisions. Note that in practice, the asymptotic size of the grid does not change because k is very small. This subdivision adds the degree of freedom necessary to eliminate the problem, making this implementation of the Marching Cubes 33 topologically correct (see Figure 4.6).

4.5.3 Issue III – Cutting-plane computation

Because this is a problem with the alternative method used in Lewiner *et al.*, the issue can be avoided by replacing the use oft_{alt} with use of the originally proposed $t_{\max}$.

4.5.4 Issue IV – Case 10

Algorithm 7 illustrates the required steps for disambiguation on case 10. We fixed the MC33 implementation by adding the lines 16-20, which in the original implementation were replaced by the result *case 10.1.1*.

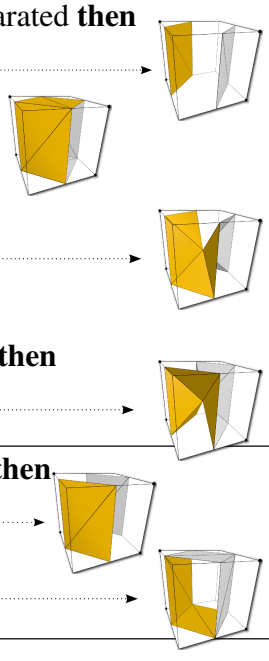
4.6 Experiments with real-world datasets

We now turn our attention to the practical impact of the topological correctness of the trilinear interpolant. For real-world datasets, the vast majority of Marching Cubes cases match the non-ambiguous configurations, namely, 1, 2, 5, 8, and 9. This means that the standard Marching Cubes will match the topology generated by both MC33 and C-MC33. Nevertheless, for some voxels, there will be topological difference in the approaches, which may result in quite different meshes.

For the sake of completeness, in this section we provide a qualitative analysis of these differences. The aneurysm dataset shown in Figure 4.9 provides an example of the differences. From left to right, Figure 4.9 shows meshes extracted with VTK Marching Cubes, MC33, and C-MC33. The VTK implementation is based on the work of Montani *et al.* [112] and does not have topological guarantees aside from consistency. These three implementations can be viewed as three distinct ways of extracting the mesh topology. Although only a handful of voxels differ among the implementations, for the aneurysm dataset the consequence is that the (largest) main brain artery appears quite different in each interpretation. Because the dataset contains several thin features, subvoxel accuracy is

Algorithm 7 Algorithm for case 10

```
1:  Positive nodes are denoted as n+
2:  if n+ are separated at top face then
3:      if n+ are separated at bottom face then
4:          if n+ at voxel diagonals are separated then
5:              Case 10.1.1
6:          else
7:              Case 10.1.2
8:          end if
9:      else
10:         Case 10.2
11:     end if
12: else
13:     if n+ are separated at bottom face then
14:         Case 10.2
15:     else
16:         if n+ at voxel diagonals joined then
17:             Case 10.1.1
18:         else
19:             Case 10.1.2
20:         end if
21:     end if
22: end if
```



required to connect the pieces of the blood vessels. As shown in the inset images in Figure 4.9, one voxel is sufficient to separate fairly large vessels.

VTK and MC33 generate more extra connected components (shown in purple) than does C-MC33. Figure 4.10 shows the difference in the number of connected components generated by VTK and C-MC33 (left) and by MC33 and C-MC33 (right) as a function of the isovalue for the aneurysm dataset. Clearly, VTK produces substantially more connected components than C-MC33 (up to 2400 more components). The differences between MC33 and C-MC33 are not as large, athough they are sufficient to disconnect important artery segments. In this example, MC33 generates more connected components than C-MC33for most isovalues, . The aneurysm dataset shows that changes in the topology of some voxels can impact the final surface. In this particular example, it is reasonable to assume that the blood vessels form a single connected component and thus that the dataset contains as few connected components as possible. Using this criterion, C-MC33 shows the best performance for most isovalues. We emphasize that the "importance" of the differences in the number of connected components ought to be measured. For instance, although in general C-MC33 produced fewer connected components, for some isovalues the number of

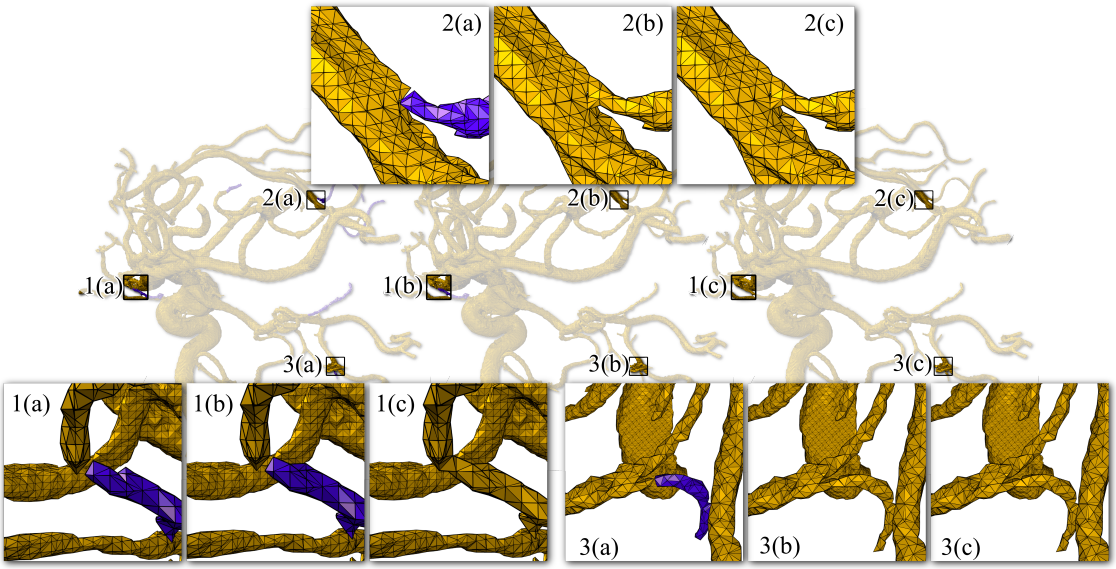


Figure 4.9. Aneurysm dataset. From left to right, the displayed isosurfaces were extracted using VTK, MC33, and C-MC33, respectively. We show the main brain artery component in yellow and the extra connected components in purple. From the images shown, it is clear that the purple components should be part of the main branch. Nevertheless, due to the implicit disambiguation in VTK and the issues in MC33, the final isosurface contains multiple components (left and middle figures). The isosurface generated using C-MC33 is shown on the right.

components extracted with C-MC33 was greater than the number extracted using MC33. As it turns out, this is due to the presence of pieces of small components disconnected from the main artery. However, because small isolated components do not disconnect large portions of the datasets, contrary to what is shown in Figure 4.9, MC33 and C-MC33 could be considered only "slightly" different. A thorough study of impact of the different approaches for extracting mesh topology is desirable but is beyond the scope of this work.

The second problem is due to the extraction of non-manifold features. The issue explained in Section 4.4.2 also pertains to in real-world datasets. Figure 4.11 shows an example of a medical dataset in which the output of MC33 implementation is a non-manifold surface. We have observed the same problem for certain isovalues of other commonly used datasets, such as the backpack and bonsai datasets. Nevertheless, in our experiments, this problem occurred rarely in the datasets tested: on average, one case of non-manifold edges was found per 10^7 evaluated voxels.

4.7 Conclusion

In this paper, we discuss in detail three issues with the Marching Cubes 33 algorithm and one non-trivial issue with its implementation. We present solutions for the issues raised and implement them into C-MC33, a topologically correct version of MC33. In addition,

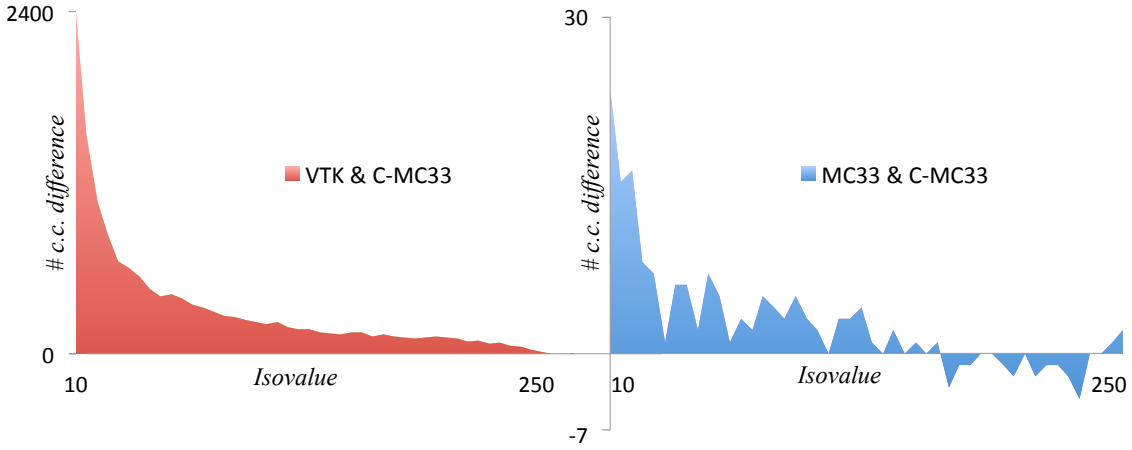


Figure 4.10. The left plot shows the difference between the number of connected components extracted by VTK implementation of Marching Cubes and the number of connected components extracted by our C-MC33 implementation. The right plot shows the difference in the number of connected components but between the MC33 and C-MC33 implementations. Negative values indicate that the C-MC33 implementation generated more connected components. Clearly, VTK generates more components that C-MC33. MC33 generates more components for most of the isovalues.

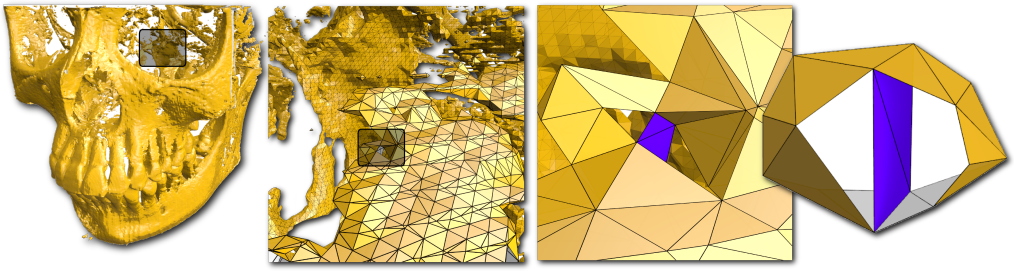


Figure 4.11. Skull dataset. The image shows a progressive zoom-in into the dataset in order to reveal non-manifold edges. The face containing the non-manifold edges is highlighted in purple. The rightmost image is an isolated version of the case shown in the dataset, with a slightly different geometry for the sake of clarity. A non-manifold edge appeared six times in total for 50 distinct isosurfaces.

we make the results of our paper reproducible so that the reader can easily study, explore, and use the results presented here for his or her own purpose.

CHAPTER 5

VERIFYING DIRECT VOLUME RENDERING ALGORITHM

Over the past decades, the visualization and graphics communities have developed a wide range of volume rendering techniques. As they are used in different disciplines of science, and thus form a basis for new scientific insights, it is essential to assess their reliability and identify errors. Furthermore, the increasing complexity of volume rendering algorithms makes the correctness of the algorithm itself as well as its potentially error-prone implementations complementary and equally important issues. Especially in areas such as medical imaging where accuracy and precision play a crucial role, a formal methodology for assessing correctness is highly desirable [72, 133]. While verification is widely adopted in different branches of computer science – see model checking [25], fuzzing [52], and convergence analysis [142] – not much work on a formalized praxis for asserting the correctness of visualization techniques has been done. In this article we present a new verification approach for direct volume rendering techniques as well as its theoretical background. We use the word verification in the same sense Babuska and Oden [3]: "verification is the process of determining if a computational model, and its corresponding numerical solution, obtained by discretizing the mathematical model (with corresponding exact solution) of a physical event, and the code implementing the computational model can be used to represent the mathematical model of the event with sufficient accuracy" [3]. The presented methodology is based on order-of-accuracy and convergence analysis [142] which we can apply after deriving the expected behavior of the algorithms under observation.

To allow the verification of volume rendering algorithms, we start with an analysis of the volume rendering integral and the most common discretization of this continuous model. This analysis gives us insight into expected behavior of the observed algorithms, which is essential to perform a verification [63]. In this sense, our main assumption, serving as a foundation for the proposed verification approach is that discretization errors of the implementations under verification should behave as the errors introduced by the

discretization of the volume rendering integral. Based on this, we can mathematically derive the expected behavior from the discretization of the volume rendering integral and verify existing implementations through convergence analysis, by comparing their actual behavior to the expected behavior. Based on the results of this comparison, we can assess the correctness of the implementation under verification. To get further insights about deviation from the expected behavior, we present an investigation of the sensitivity of this method. Thus, we can demonstrate that our methodology is capable of increasing the confidence in volume rendering algorithms. To our knowledge, the proposed approach is the first step towards the verification of DVR algorithms. Thus, it can be seen as an important contribution towards a formal verification methodology of volume rendering techniques [139]. The main contributions of this article are:

- we derive the theoretical foundations necessary for verifying volume rendering with order-of-accuracy and convergence analysis. We analyze the volume rendering integral and its discretization to derive an algorithm's expected behavior when being subject to parameter changes.

- we explain how to exploit this theoretical foundation to perform a practical verification of implemented volume rendering algorithms, such that it can be easily used for the verification of existing volume rendering frameworks;

- we discuss the limitations of the proposed concepts by analyzing frequently occurring errors and by documenting those errors we could identify when applying the presented methodology to two widely used volume rendering frameworks, VTK [151] and Voreen [109] (see Figure 5.1).

5.1 Related work

Critical decisions in fields such as medical imaging often rely on images produced by volume rendering algorithms, where it is of utmost importance that the results are correct [32]. The multitude of algorithms components and their interactions make this guarantee a challenge. As a consequence, many authors focus on specific aspects of the problem such as numerical aspects of the evaluation of the volume rendering integral, shading, transfer functions, and interpolation schemes. The quality of volume rendering has always been of central interest to the community, and relying on visual inspection is a common practice. Meissner *et al.* [107] evaluate volume rendering techniques using the human visual system as a reference while, more recently, Smelyanskiy *et al.* [157] present a

domain expert guided comparison scheme. While those approaches are valuable, the need for a more systematic evaluation is discussed in several papers [51, 66, 67, 72]. See Pommert and Höhne [133, 134] for a survey.

Among several aspects to consider in the correctness of volume rendering algorithms, one of the most important is the approximation of the volume rendering integral. The solution with linearly interpolated attributes is presented by Williams and Max [183], with further discussions on its numerical stability by Williams *et al.* [184]. Interpolant approximations and errors [37, 110, 111, 124], gradient computation [174] and opacity correction [86] are also the subject of analysis with regard to numerical accuracy. The idea of *pre-integration* enabled high-quality, accurate and efficient algorithms using graphics hardware [40, 81, 141]. Similarly, VTK currently uses partial pre-integration, in particular for unstructured grids [115]. Note that although there has been work on high-order and high accuracy volume rendering, to the best of our knowledge none of these approaches attempted to evaluate the convergence rate of the standard discretization process of the volume rendering integral, thus providing weaker mechanism to evaluate whether or not the mathematical foundations of the algorithms have been implemented in a correct manner.

The use of a verification framework has only recently been discussed in scientific visualization, despite the vast literature on verification in computer science. Globus and Uselton [51] first pointed out the need to verify not only visualization algorithms but also their implementations, and Kirby and Silva suggested a research program around verification [72]. The verification of isosurface algorithms is discussed by Etiene *et al.* [41, 42], where a systematic evaluation identified and corrected problems in several implementations

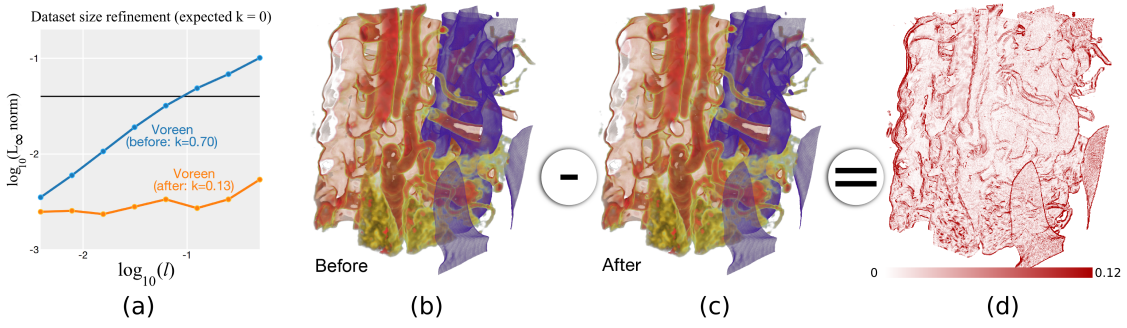


Figure 5.1. (a) shows the result of our verification procedure for dataset refinement. The blue line corresponds to the initial behavior, which deviates from the expected slope (solid dark line). After fixing the issues, we obtain the orange curve, with a slope closer to the expected one (denoted by k). (b) and (c) show a human torso, displaying the blood vessels and the spine, before and after our changes. (d) shows the difference between (b) and (c).

of isosurface extraction techniques. Zheng *et al.* [189] address CT reconstruction and interpolation errors in direct volume rendering algorithms using a verifiable framework based on projection errors. In contrast, our work focuses on the verification of the final image produced through direct volume rendering.

5.2 Verification

Before presenting our verification procedure, let us consider four of the techniques used for code verification in computational science [142]: *expert judgment*, a procedure in which a field expert determines if the output of an implementation is correct by evaluating the results; *error quantification*, which is the quantification of the discretization errors when compared to an analytical solution, a benchmark solution or some ground-truth; *convergence analysis*, a procedure in which one evaluates if the discretization errors converge to zero as a function of some parameter; and *order-of-accuracy*, a procedure where one evaluates if the discretization errors decrease according to the expected rate. In this list, the expert judgment is the least rigorous test, followed by error quantification and convergence analysis. Order-of-accuracy is widely recognized as the most rigorous code verification tool [3, 76, 139, 142]. In this paper, we focus on the last two methods, namely, convergence analysis and order-of-accuracy. Before we dive into these methods, let us first consider some of the limitation of the expert analysis and error quantification.

In visualization, expert analysis and error quantification are, to the best of our knowledge, the only two verification tools previously employed for verification of volume rendering techniques [107, 110, 157]. Whereas it is easy to envision situations where an expert may fail to predict a code mistake, it is more difficult to see when error quantification fails. We devise the following experiment to understand potential limitations of both approaches. We artificially introduced a code mistake in a volume rendering implementation: the trilinear interpolation was changed from $p(x,y,z) = Axyz + \underline{Bxy(1-z)} + \ldots$ to $p(x,y,z) = Axyz + \underline{Axy(1-z)} + \ldots$. We then used this implementation to render an image whose analytical solution is known. Finally, we compute the maximum error between the rendered and the analytical solution, which in this case is 3.6×10^{-3}. How can one decide if this value is good enough? Does the sampling distance d or the input scalar field $s(x,y,z)$ give us enough data to make an informed decision? In this particular case, the correct interpolant generates an image with maximum error of 3.4×10^{-3}: the two images are very similar by this metric. Also, it may be challenging even for an expert to notice such small deviation, as shown in Figure 5.2. On top of this, the maximum errors for another code

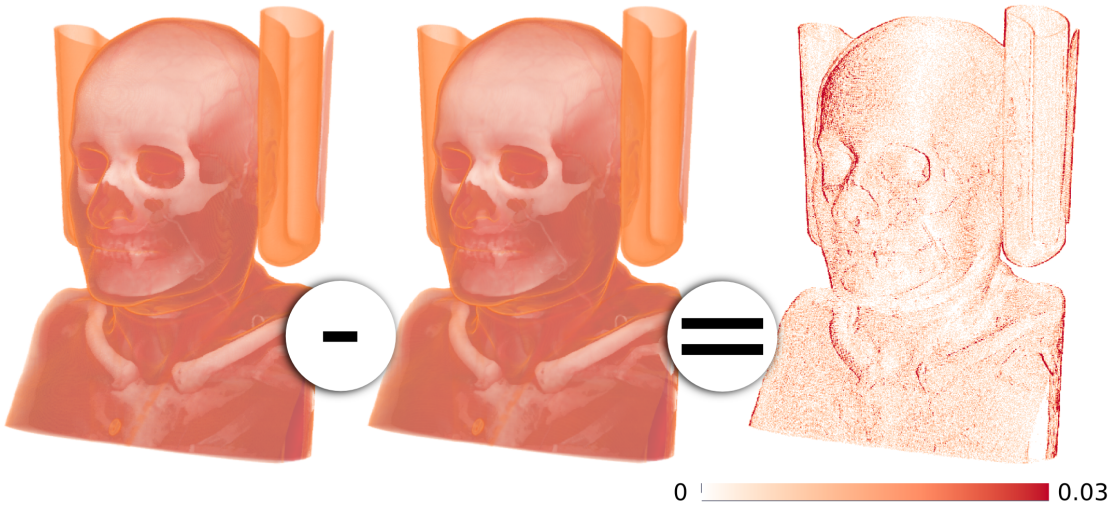


Figure 5.2. Left: the volume rendering of the torso dataset with incorrect trilinear interpolant. Middle: Same dataset with the correct interpolant. Right: image shows the difference between them.

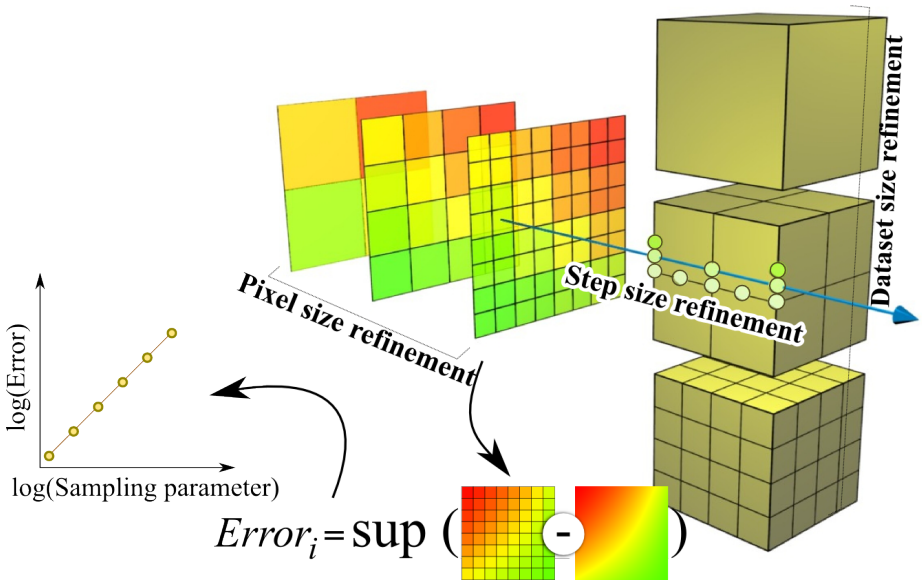


Figure 5.3. Our verification procedure works by evaluating discretization error during refinement of one of three sampling parameters.

mistake could be even smaller. (We point out that this particular case can be uncovered by "playing around" with the data or other *ad hoc* methods. The goal of this example is to show that error quantification can also fail to predict code mistakes, even for a severe bug.) On the other hand, we will have enough information to make such a decision if one observes how errors behave when input parameters change, instead of quantifying them from one image. The convergence and order-of-accuracy tests work in this way, and they are the focus of this paper.

We advocate the use of convergence and order-of-accuracy verification not as a replacement but as an extension of the current testing pipeline. Note that these are not the only approaches for assessing correctness of computer code. As mentioned before, verification is well-developed in computer science [25, 43, 52, 187].

We apply verification in the spirit of Babuska and Oden's procedure, which we summarize in Figure 5.3 [3]. It starts with a mathematical evaluation of the expected convergence of the volume rendering integral (Section 5.4). The result of this step is the asymptotic error according to some discretization parameter (step size, dataset size, or pixel size). Then, we use the volume rendering implementation under verification to generate a sequence of images by successive refinement of one of the discretization parameters. Next, we compute the observed discretization errors by comparing these images against a reference – an analytical solution, if one is available, or one of the rendered images. Finally, we compare the sequence of observed outputs against expected errors to evaluate if expected and observed convergence match (Sections 5.5 and 5.6).

5.3 Discretization errors

In this section we present the mathematical model used in volume rendering algorithms and its *expected behavior*, which we write in terms of the errors involved in each discretization step. Let us assume the well-known *low albedo emission* plus *absorption* model [102]. The volume rendering integral (VRI) I, as described by Engel *et al.* [40], is:

$$\begin{aligned} I(x,y) &= \int_0^D C(s(\mathbf{x}(\lambda)))\tau(s(\mathbf{x}(\lambda))) \\ &\qquad \times \exp\left(-\int_0^\lambda \tau(s(\mathbf{x}(\lambda')))\mathrm{d}\lambda'\right)\mathrm{d}\lambda, \end{aligned} \tag{5.1}$$

where D is the ray length, $C(s(\mathbf{x}(\lambda)))$ is the reflected/emitted light, $\tau(s(\mathbf{x}(\lambda)))$ is the light extinction coefficient, $s(\mathbf{x}(\lambda))$ is the scalar value at position $\mathbf{x}$ in the ray parameterized by λ. There are three natural ways to discretize the equation. We will generate progressively denser ray sampling (by refining the integration *step size*), progressively larger datasets (by

refining the *size of the voxel in the dataset*), and progressively higher-resolution images (by refining the *pixel size in the final image*). Each of these three variables will introduce errors that will appear in the discretization of the VRI. In the following section, we discretize the VRI using the most common approximation in literature.

5.3.1 Errors due to step size refinement

In this section, we are interested in the errors generated by successive ray step refinements (see Figure 5.4). We first generate a sequence of images I_i using progressively smaller step size. We then evaluate the error E_i. Equation (5.1) is commonly discretized using traditional Riemann sums for numerical integration:

$$\int_0^D f(s(\mathbf{x}(\lambda)))\mathrm{d}\lambda = \sum_{i=0}^{n-1} f(s(\mathbf{x}(id)))d + O(d), \tag{5.2}$$

where n is the number of sub-intervals and $d = D/n$. The proof of linear convergence follows from Taylor expansion of the integrand over small intervals d. Other methods are available and they provide different convergence rates. For instance, the Trapezoid method is a 2nd order method on the integral of f.

In the case of the VRI, we approximate not only the outer integral but also the integrand $T(s(\mathbf{x}(\lambda))) = \exp\left(-\int_0^\lambda \tau(s(\mathbf{x}(\lambda')))\mathrm{d}\lambda'\right)$. Moreover, T requires two approximations: $e^{t(\lambda)}$ and the inner integral. Before we derive the convergence rate for the VRI, let us first evaluate the convergence of T. Throughout the text, we assume that all transfer functions are smooth, *i.e.,* $C(s), \tau(s) \in C^\infty$. Although this is not the case in practice, this restriction is useful for convergence evaluation and verification purposes.

5.3.1.1 Approximation of $T(\lambda)$

Let $T(\lambda) = T_\lambda = e^{-t(\lambda)}$, where $t(\lambda) = \int_0^\lambda \tau(\lambda')\mathrm{d}\lambda'$, and λ parameterizes a ray position. We will first approximate $t(\lambda)$ and then $T(\lambda)$. Typically, the integral is solved by using Riemann sums. In the following, $d = D/n$ is the ray sampling distance, D is the ray length and n is the number of sub-intervals along the ray:

$$\int_0^\lambda \tau(\lambda')\mathrm{d}\lambda' \quad = \quad \sum_{j=0}^{i-1} \tau(jd)d + O(d) \tag{5.3}$$

where $\lambda = id$. Using Equation (5.3):

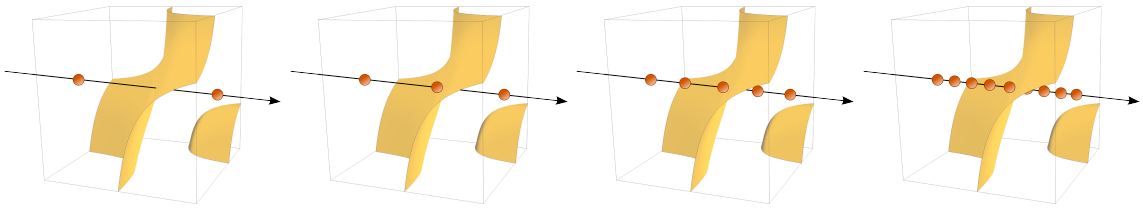


Figure 5.4. Step size refinement. The figure shows an isosurface of the trilinear function defined on the volume.

$$T(\lambda) = \exp\left(-\int_0^{\lambda} \tau(\lambda')\mathrm{d}\lambda'\right) \tag{5.4}$$

$$= \exp\left(-\sum_{j=0}^{i-1} \tau(jd)d + O(d)\right) \tag{5.5}$$

$$= \left(\prod_{j=0}^{i-1} \exp\left(-\tau(jd)d\right)\right) \exp\left(O(d)\right). \tag{5.6}$$

Let us define $\tau_j = \tau(jd)$. We start with a Taylor expansion of $\exp\left(O(d)\right)$:

$$T_\lambda = \left(\prod_{j=0}^{i-1} \exp\left(-\tau_j d\right)\right)\left(1 + O(d)\right). \tag{5.7}$$

In the equation above, we assume that high order terms are negligible, and thus the dominant error is $O(d)$:

$$T_\lambda = \left(\prod_{j=0}^{i-1} \exp\left(-\tau_j d\right)\right)\left(1 + O(d)\right) \tag{5.8}$$

$$= \prod_{j=0}^{i-1} \exp\left(-\tau_j d\right) + \prod_{j=0}^{i-1} \exp\left(-\tau_j d\right) O(d) \tag{5.9}$$

Part I Let us focus on the second term in the right hand side of Equation (5.9). The first observation is that it contains only approximation errors, which means that we are interested only in its asymptotic behavior. Let us expand it using first-order Taylor approximation and use the fact that $\tau_j d = O(d)$:

$$\prod_{j=0}^{i-1} \left(1 - O(d)\right) O(d) = \left(1 + O(d)\right)^i O(d), \tag{5.10}$$

where the change in the sign is warranted because the goal is to determine the asymptotic behavior. For $i = 1$, only one step is necessary for computing the volume rendering integral along the ray, and the previous equation will exhibit linear convergence. Nevertheless, in

the general case, the numerical integration requires multiple steps, hence errors accumulate, and the convergence may change. Thus we set $i = n$. Knowing that $(1 + O(d))^n = O(1)$ (see Appendix B) and inserting Equation (5.10) into Equation (5.9) we obtain:

$$T_\lambda = \prod_{j=0}^{i-1} \exp(-\tau_j d) + O(d)O(1), \tag{5.11}$$

and the Taylor expansion of the first term yields:

$$T_\lambda = \prod_{j=0}^{i-1} \left(1 - \tau_j d + O(d^2)\right) + O(d). \tag{5.12}$$

Part II We now show that the first term on the right side of Equation (5.12) also converges linearly with respect to d. In the course of this section, we omit the presence of the term $O(d)$ in Equation (5.12) for the sake of clarity. Let us define the set K as the set of indices j for which $1 - \tau_j d = 0$. The size of K is denoted as $|K| = k$. We also define N as the set of indices j for which $1 - \tau_j d \neq 0$, and $|N| = i - k$. Equation (5.12) can be written as:

$$T_\lambda = \left(\prod_{j\in N} 1 - \tau_j d + O(d^2)\right)\left(\prod_{j\in K} O(d^2)\right) \tag{5.13}$$

$$= \left(\prod_{j\in N} 1 - \tau_j d + O(d^2)\right) O(d^{2k}). \tag{5.14}$$

Because $1 - \tau_j d \neq 0$ for $j \in N$:

$$T_\lambda = \left(\prod_{j\in N} (1 - \tau_j d)\left(1 + \frac{O(d^2)}{1 - \tau_j d}\right)\right) O(d^{2k}) \tag{5.15}$$

From the definition of big O notation, $1/(1 - \tau_j d) = O(1)$, hence:

$$T_\lambda = \left(\prod_{j\in N} (1 - \tau_j d)\left(1 + O(1)O(d^2)\right)\right) O(d^{2k}) \tag{5.16}$$

$$= \left(\prod_{j\in N} (1 - \tau_j d)(1 + O(d^2))\right) O(d^{2k}) \tag{5.17}$$

$$= \left(\prod_{j\in N} 1 - \tau_j d\right) (1 + O(d^2))^{i-k} O(d^{2k}). \tag{5.18}$$

In real world implementation, $k \neq 0$ implies that at least one of the terms $1 - \tau_j d = 0$. Hence the code accumulating the value of T

```
T = T * (1 - t_j * d)
```

will invariably return `T = 0`. This can also be seen in our theoretical analysis. For $k \neq 0$, the whole right hand side of Equation (5.18) *is* the approximation error. The larger k is – *i.e.* the more zeroes in the product of Equation (5.18) – the faster the sequence converges to 0 because of the $O(d^{2k})$ factor. So, when $k \neq 0$, we will have a high order approximation of $T_\lambda = 0$. Nevertheless, because we want to recover the approximation errors for the general case ($T_\lambda \neq 0$), we set $k = 0$ in Equation (5.18), and $i = n$ (for the same reasons as previously stated):

$$T_\lambda = \left(\prod_{j=0}^{n-1} 1 - \tau_j d \right) (1 + O(d^2))^n. \tag{5.19}$$

Using the fact that $(1 + O(d^2))^n = 1 + O(d)$ and $(1 + O(d))^n = O(1)$ (see Appendix B):

$$T_\lambda = \left(\prod_{j=0}^{n-1} 1 - \tau_j d \right) (1 + O(d)) \tag{5.20}$$

$$= \left(\prod_{j=0}^{n-1} 1 - \tau_j d \right) + O(d) \left(\prod_{j=0}^{n-1} (1 + O(d)) \right) \tag{5.21}$$

$$= \left(\prod_{j=0}^{n-1} 1 - \tau_j d \right) + O(d)(1 + O(d))^n \tag{5.22}$$

$$= \left(\prod_{j=0}^{n-1} 1 - \tau_j d \right) + O(d)O(1) \tag{5.23}$$

$$= \left(\prod_{j=0}^{n-1} 1 - \tau_j d \right) + O(d) \tag{5.24}$$

We finish our derivation by adding the previously omitted $O(d)$ term:

$$T_\lambda = \left(\prod_{j=0}^{n-1} 1 - \tau_j d \right) + O(d) + O(d) \tag{5.25}$$

$$= \left(\prod_{j=0}^{n-1} 1 - \tau_j d \right) + O(d) \tag{5.26}$$

5.3.1.2 Approximation of the outer integral

Let $\tilde{T}_i$ be the approximation of $T(\lambda_i)$. We write $T(\lambda_i) = T_i = \tilde{T}_i + O(d)$, and $C_i = C(id)$. In typical volume rendering implementations, the outer integral is also approximated using a Riemann sum. Thus:

$$I(x,y) = \sum_{i=0}^{n-1} C(id)\tau(id)T_i d + O(d) \tag{5.27}$$

$$= \sum_{i=0}^{n-1} C_i\tau_i d\left(\tilde{T}_i + O(d)\right) + O(d) \tag{5.28}$$

$$= \sum_{i=0}^{n-1} C_i\tau_i\tilde{T}_i d + \sum_{i=0}^{n-1} C_i\tau_i dO(d) + O(d). \tag{5.29}$$

Because both τ_i and C_i are bounded, one can write $C_i\tau_i dO(d) = O(d^2)$ and $\sum_i O(d^2) = nO(d^2) = \frac{D}{d}O(d^2) = O(d)$. The above equation can be re-written as:

$$I(x,y) = \sum_{i=0}^{n-1} C(id)\tau(id)d\tilde{T}_i + O(d). \tag{5.30}$$

We have now showed that the dominant error when considering step size in the VRI is of order $O(d)$. In other words, when decreasing the step size by half, the error should be reduced by half.

5.3.1.3 Numerical integration techniques

The interplay between the approximation errors of the inner and outer integrals is non-trivial, and here we show a simple numerical example. Table 5.1 shows the results of the effects of different integration methods for the inner and outer integrals along a single ray. We simulate the integration along a single ray to compute these quantities numerically. For this experiment, we assume: $x \in [0, D]$, $\tau(s(x)) = \cos(s(x))$, $C(s(x)) = \sin(s(x))$, $s(x) = x$ and thus the solution for the VRI is $I = 1 - \exp(-\sin(D))(\sin(D) + 1)$. To evaluate the effects of the discretization errors of the integrals, we further assume that $\exp(x)$ does not introduce errors. The computation of the convergence rate is detailed in Section 5.4. The results shown in Table 5.1 suggest that one needs to improve the accuracy of *both* the inner and outer integrals to obtain high-order methods.

5.3.2 Errors due to dataset refinement

For the purposes of this paper, we assume that no additional errors will be created during the refinement of the scalar field. Hence, we need to find an interpolation function,

Table 5.1. Effects of the different integration methods.

		Outer integral	
		Riemann	Trapezoid
Inner integral	Monte Carlo	$O(d^{0.43})$	$O(d^{0.58})$
	Riemann	$O(d^{0.99})$	$O(d^{0.98})$
	Trapezoid	$O(d^{1.01})$	$O(d^{2.00})$

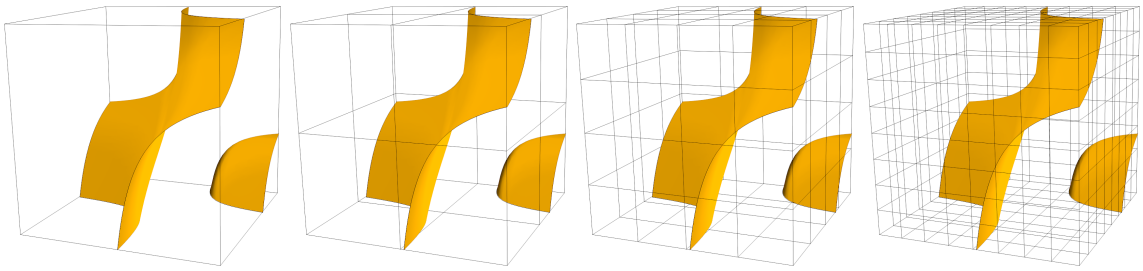


Figure 5.5. Isosurface of a randomly generated scalar field defined in different resolutions. The (piecewise) trilinear surface is the same regardless of the grid size.

that fulfills the so called two-scaling property. Fortunately, B-splines fulfill the two-scale property [173] and we will pick the linear B-spline, which results in the well-known tri-linear interpolator. Care must be taking on the refinement step. In this paper we will pick a refinement factor of two, which simplifies to a simple averaging of the nearest neighbours. The errors introduced so far remain unchanged:

$$I(x,y) = \sum_{i=0}^{n-1} C(\tilde{s}(\mathbf{x}_i))\tau(\tilde{s}(\mathbf{x}_i))d\tilde{T}(\tilde{s}(\mathbf{x}_i)) + O(d). \tag{5.31}$$

The fact that no error due to grid resolution is introduced into Equation (5.31) can be interpreted as an "infinity order of convergence" on the grid size m, $O(m^{-\infty})$; *i.e.*, an instantaneous convergence to zero error. Nevertheless, Equation (5.31) still contains an error due to ray sampling $O(d)$, and others not considered in this work (such as rounding errors, finite precision, etc.). These fixed error sources will be revealed in the approximation of the VRI and remain (mostly) unchanged when refining dataset size. Hence, even though no errors are introduced, one still expect constant error on the approximation of the VRI, mainly due to the $O(d)$ term. Thus, from now on, we will say that the expected order of accuracy for the grid size *test* is constant, even though the convergence rate *with respect to* m is $O(m^{-\infty})$.

5.3.3 Errors due to pixel size refinement

The final source of errors we investigate comes from the finite number of rays sent into the image. One way of quantifying these errors is to create a sequence of images of progressively higher resolution, and then examine the supremum of the difference between values of the finite approximations of the volume-rendered image and the true solution. In this section, we assume that the derivatives along image axes of the volume rendering integral exist.

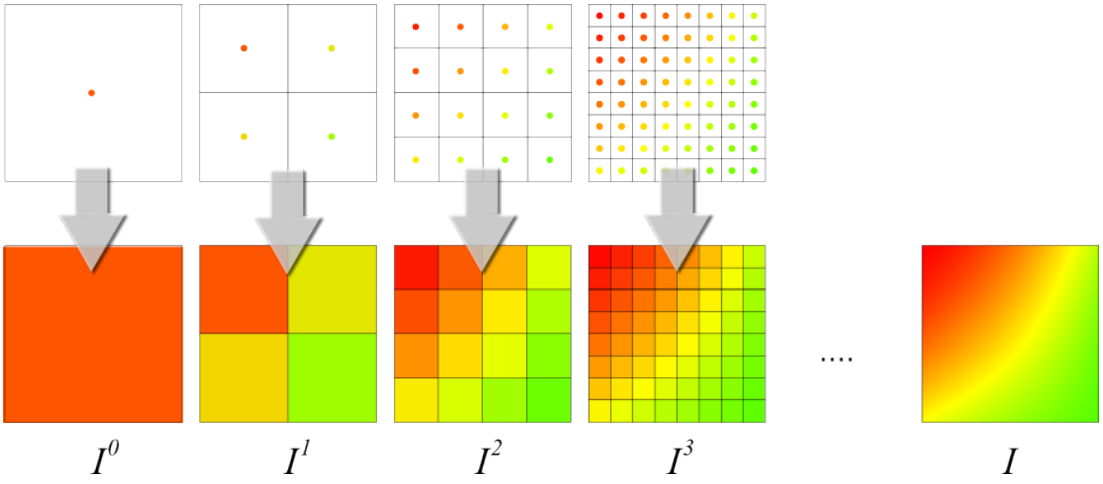


Figure 5.6. Pixel refinement. Typically, the VRI is evaluated only at pixel centers (top). The value at the center is then interpolated in the domain defined by the pixel size (bottom) using nearest-neighbor interpolation to obtain $\tilde{I}^i$. In a correct implementation, as i increases, $\tilde{I}^i$ approaches the true solution I.

Denote the true volume-rendered image as $I(x, y)$. The approximation is constructed by sampling $I(x, y)$ in a finite subset of the domain (in our case, a square lattice of increasing resolution). At a level of detail j, $\tilde{I}^j(x, y)$ denotes the nearest-neighbor interpolation of the sampled values, and the error is measured as $E_j = \sup_{(x,y)\in[0,1]^2} |I(x, y) - \tilde{I}^j(x, y)|$. Effectively, this procedure assigns to the entire square pixel the value sampled in its center, but evaluates the error *over the entirety of the pixel values*. Figure 5.6 illustrates the process.

We use the notation $I = I(x, y)$, $I_i^j = I^j(x_i, y_i)$, and $\delta_i = (x, y)-(x_i, y_i)$. In what follows, the Taylor expansion assumes a fixed level of detail j. We omit the superscript j for the sake of clarity. Let us write the values of I as a Taylor series expansion ($\mathbf{H}_i = \mathbf{H}(x_i, y_i)$ and is the Hessian and I_i^x and I_i^y are the partial derivatives of I_i at (x_i, y_i)):

$$
\begin{aligned}
I &= \tilde{I}_i + \nabla I_i^T \delta_i + \frac{1}{2}\delta_i^T \mathbf{H}_i \delta_i + \cdots && (5.32)\\
&= \tilde{I}_i + O(\nabla I_i^T \delta_i) && (5.33)\\
&= \tilde{I}_i + O((I_i^x, I_i^y)^T (x - x_i, y - y_i)), && (5.34)
\end{aligned}
$$

$\tilde{I}_i$ is a nearest-neighbor reconstruction from a square lattice for the pixel (x_i, y_i) and at a given level j. In the regime where the Hessian terms are negligible, the dominant errors (and hence the supremum of the difference) occur when $(x - x_i, y - y_i) = (h, h)$, where h is half the pixel size. Thus:

$$
\begin{aligned}
I &= I_i + O(h) && (5.35)\\
&= \sum_{i=0}^{n-1} C(\tilde{s}(\mathbf{x}_i))\tau(\tilde{s}(\mathbf{x}_i))\tilde{T}d(\tilde{s}(\mathbf{x}_i))) + \\
&\qquad + O(h) + O(d). && (5.36)
\end{aligned}
$$

As can be seen, the error in pixel approximation decays linearly with the pixel size. Equation (5.36) contains all the errors we use for verification purposes and it will be the base for our analysis in Section 5.4.

In practice, often the final image is not a smooth function. However, our goal is not to provide a characterization of discretization errors that can be used in any setting, but instead one that can be used for verification purposes. Therefore, to use the analysis outlined above, one must manufacture scalar field and transfer functions which yield a smooth function (cf. Section 5.5).

5.4 Convergence computation

The heart of our method is the evaluation of the discretization errors. Once we have the discretization errors, we can evaluate the order-of-accuracy and convergence rate. The error computation and analysis will proceed differently depending on whether an analytical solution for the VRI is available or not. We highlight that previous frameworks for verification of visualization algorithms could benefit from the fact that analytical solutions can be easily constructed [42]. For the case of VRI, this is no longer true and therefore we should not count only on known solutions. We describe two ways in which to proceed with the convergence analysis. First, we will show how to calculate errors using a known solution, and then how to do so using an unknown solution.

5.4.1 Numerical errors using a known solution

When a solution $F(x, y)$ for the VRI is known, the procedure is equivalent to the Method of Manufactured Solutions [3]. In the previous section, we have shown that the solution F can be written as:

$$F(x, y) = I(x, y) + O(r^k) = I(x, y) + \beta r^k + \text{HOT}, \tag{5.37}$$

where I is the approximated image, r is the discretization parameter (step size, grid size, or pixel size) and $\beta \in \mathbb{R}$ is a constant, multiplicative factor that is not a function of the dataset. An important assumption is that HOT, or "higher order terms", are small enough so that it does not affect the convergence of order $k \in \mathbb{R}$, *i.e.*, high order derivatives of F must have negligible impact in the asymptotic convergence of I [142]. This formulation

implies that not all solutions F are suitable for verification purposes, only those for which HOT is negligible. In addition, integration methods whose approximation errors cannot be written as shown, cannot be compared by only evaluating k, as we propose next. The expected value of k for the cases of step size and pixel size convergence is $k = 1$ whereas $k = 0$ for grid size. This implies that the pixel intensity converges to the true solution with "speed" determined by k and thus the error can be written as:

$$e(x, y) = I(x, y) - F(x, y) \approx \beta r^k. \tag{5.38}$$

One can evaluate the convergence for all pixels in the image using L_2, L_∞, or other norms. Henceforth, we adopt the L_∞ because it provides a rigorous way of evaluating errors: it tells us that the maximum image error should decay at the same rate k. Mathematically, the error is then:

$$E = \sup_{x,y}(e(x, y)) = \sup_{x,y}(|I(x, y) - F(x, y)|) = \beta r^k. \tag{5.39}$$

We will denote individual images (and the respective errors) by a subscript i. For each image I_i, we first calculate the supremum of the absolute difference $\sup_{x,y}(|F(x, y) - I_i(x, y)|)$. Then, we compute the observed convergence rate k by taking logarithms of both definitions of E and solving the resulting equations for $\log(\beta)$ and k in a least-squares sense:

$$\begin{aligned} \log E_i &= \log \sup_{x,y}(|F(x, y) - I_i(x, y)|) \\ &= \log(\beta) + k \log(r_i). \end{aligned} \tag{5.40}$$

The system of equations has as many equations as the number of images and calculated errors. We note that the solution $F(x, y)$ cannot always be computed analytically [102]. In the general case, we need an alternative method for determining the error.

5.4.2 Numerical errors using an unknown solution

In the case of an unknown solution, using a numerical approximation in a high-precision context to compute a reference image is a valid approach for verification [79]. The main disadvantage is that it might mask errors which appear in the reference image itself. Our slightly different approach requires neither an analytical solution nor a numerical approximation, but still retains a high sensitivity to errors. Suppose we want to verify the convergence of a sequence of images I_i with $r_{i+1} = cr_i$, where $c \in (0, 1)$ is a constant

factor. As we have seen in the previous section, the approximation for the solution F at resolution i and $i+1$ can be written respectively as:

$$\begin{aligned} F(x,y) &= I_i(x,y) + O(r_i^k) \\ &= I_i(x,y) + \beta r_i^k + \text{HOT}, \qquad (5.41) \\ F(x,y) &= I_{i+1}(x,y) + O(r_{i+1}^k) \\ &= I_{i+1}(x,y) + \beta r_{i+1}^k + \text{HOT}. \qquad (5.42) \end{aligned}$$

Again, we assume that HOT are negligible. Now, we subtract Equation (5.42) from (5.41) to eliminate the unknown F:

$$\begin{aligned} 0 &= (I_{i+1}(x,y) + \beta r_{i+1}^k) - (I_i(x,y) + \beta r_i^k) \qquad (5.43) \\ 0 &= I_{i+1}(x,y) - I_i(x,y) + \beta r_{i+1}^k - \beta r_i^k. \qquad (5.44) \end{aligned}$$

Thus, the convergence order k can be computed by evaluating the errors involved in the subtraction consecutive images:

$$\begin{aligned} e_i(x,y) = I_{i+1}(x,y) - I_i(x,y) &= -\beta r_{i+1}^k + \beta r_i^k \qquad (5.45) \\ &= \beta(1-c^k) r_i^k. \qquad (5.46) \end{aligned}$$

As before, we use L_∞ norm to compute the maximum error among all pixels:

$$\begin{aligned} E_i &= \sup_{x,y}(e_i(x,y)) \\ &= \sup_{x,y}(|I_{i+1}(x,y) - I_i(x,y)|) = \beta(1-c^k) r_i^k. \qquad (5.47) \end{aligned}$$

Thus the observed convergence is again computed by taking logarithms of both sides. We then write $y = \log \beta(1-c^k)$ to hide the dependency of the term in k and determine y and k via least-squares:

$$\begin{aligned} \log E_i &= \log \beta(1-c^k) r_i^k \qquad (5.48) \\ &= \log \beta(1-c^k) + k \log r_i \qquad (5.49) \\ &= y + k \log r_i. \qquad (5.50) \end{aligned}$$

The case for grid size refinement is slightly different. Because our formulation does not introduce any error due to grid refinement, we cannot evaluate E_i in terms of r_i. In this case, one should expect only errors due to ray sampling and other error sources that are not considered in this analysis (such as rounding errors, fixed-point precision, etc). Therefore, we expect to see approximately small and constant errors E_i between successive images

when grid size is refined, as can be seen in Figure 5.7 (c), (f), and (i). This implies that expected slope is $k = 0$.

Equation (5.50) shows us how to compute the convergence rate using only the images obtained from the VRI approximation and consequently avoiding any bias and/or limitations introduced by simple manufactured solutions or numerical approximations using reference images. We have generated sequences of images based on the refinements in the following section. The steps are shown in Algorithm 8.

Algorithm 8 A simple algorithm for verification via step size, dataset size or pixel size.

VERIFICATION PROCEDURE($G, \tau(s), d_0, m_0, h_0, \rho$)

1 ▷ Let G be the scalar field
2 ▷ Let $\tau(s)$ be a transfer function
3 ▷ Let d_0, $m_0 \times m_0 \times m_0$ and h_0 be the initial step size, dataset size and pixel size respectively
4 ▷ Let $\rho \in \{\mathsf{step}, \mathsf{dataset}, \mathsf{pixel}\}$
5 $F_0 \leftarrow$ VOLUMERENDERING($G, \tau(s), d_0, m_0, h_0$)
6 **for** $i \leftarrow 1$ **to** #tests
7 **do** REFINE($d_i, m_i,$ or h_i depending on ρ)
8 $F_i \leftarrow$ VOLUMERENDERING($G, \tau(s), d_i, m_i, h_i$)
9 **if** there is an analytical solution I:
10 **then** $E_i = \sup_{x,y} |I(x, y) - F_i(x, y)|$
11 **else** $E_i = \sup_{x,y} |F_{i-1}(x, y) - F_i(x, y)|$
12 Linear regression of E_i using Equations (5.40) or (5.50)

5.5 Application examples

We present the results of applying our verification framework to two mature and widely used libraries, namely VTK and Voreen. We stress that the goal here is first to show that our verification technique is very sensitive to changes that cause the output image to deviate from the correct solution; secondly, it is very easy to apply and thus can help developers and practitioners to gain confidence in their implementations.

5.5.1 Implementations under verification

In this section we show the implementations under verification.

VTK The VTK library provides several implementations of well-known volume rendering techniques. In our tests we included two modules from version 5.6.1: `vtkVolumeRayCastMapper` (RCM) and `vtkFixedPointVolumeRayCastMapper` (FP). The RCM module

accepts as input scalar fields with 8- or 16-bit precision and internal computations are performed with single or double precision. FP accepts input datasets with up to 32 bits of precision but it uses 15-bit fixed-point arithmetic internally. Both techniques use back-to-front compositing. We have also modified the VTK source to capture 15 bit and 32 bit precision images for FP and RCM respectively.

Voreen As opposed to the tested modules in VTK, Voreen uses the graphics processing unit (GPU) and front-to-back compositing for its implementations. From the ray casting processors available within Voreen, we have chosen the `SingleVolumeRaycaster`, which is the standard processor in most Voreen workspaces. At the time of writing, version 2.6.1 is the latest, and the one we verified. We made minor modifications to the code so that floating point data of the format Nearly Raw Raster Data NRRD [70] could be imported and smaller step sizes could be used.

5.5.2 System setup

The grid lies in the domain $[0, 2]^3$ for VTK and $[0, 1]^3$ for Voreen. The scalar values at grid nodes are chosen from a uniform random distribution. The camera is centered at the xy plane and aims along the z axis. We did not include shading, since that gives a more complex VRI. To verify shaded results, a different theoretical analysis is necessary. The images can be generated using both perspective and parallel projections. We only use post-classification, which simplifies the analysis. In addition, we assume an identity opacity transfer function (that is, the opacity is exactly equal to the sampled scalar). We do this because for every pair of scalar field and opacity transfer function, there is another scalar field (which admittedly need to be of finer resolution) that, when combined with the identity transfer function, gives the same result. The function composition arising from volume classification can increase the high-frequency content of a volume [4], and a full treatment of the impact of arbitrary transfer functions on the convergence of the integral remains a topic for future explorations. In addition, this assumption enabled much of the theoretical analysis that would not be possible otherwise, while still being stringent enough to uncover issues in the implementations.

To apply verification via step size refinement, we start with $d_0 = \frac{1}{2}$ and a refinement factor of half, $d_{i+1} = \frac{1}{2}d_i$. We use a dataset of size 2^3 since we have experienced that low resolution grids with random scalar fields are effective at stressing the code for debugging purposes.

Let l be the cell size. For verification via dataset refinement we start with 2^3 grid nodes, and we refine grid cells until we reach 513^3 nodes, corresponding to cell sizes $l_{i+1} = \frac{1}{2}l_i$.

Step size is fixed at $d = 10^{-2}$. This is done to evaluate the effects of discretization errors due only to grid refinement.

For verification via pixel size refinement, we start by generating images with 32^2 pixels using the implementation under verification, and then continue to refine pixel size until we reach 1024^2 pixels. The pixel size h is refined according to $h_{i+1} = \frac{1}{2}h_i$. The errors are computed taking the difference between the rendered image and an analytical solution. In this case, we use an analytical solution for the volume rendering integral in the domain $[0, 1]^2$. We assume the following: $s(x, y, z) = z\cos(xy)$, $\tau(s) = \sin(s)$, $\mathbf{x}(\lambda) = (x, y, \lambda)$, $C(s) = 1$ and ray length $D = 1$. The analytical solution is then:

$$I(x, y) = 1 - \exp\left(\frac{\cos(\cos(xy))}{\cos(xy)} - \frac{1}{\cos(xy)}\right). \tag{5.51}$$

The dataset size used is 513^3, and the step size is set at $d = 10^{-5}$ to mitigate sampling errors. Both step and dataset size are fixed to only evaluate errors due to pixel size refinement.

For VTK, we also have the following setup: no auto adjustment of the step size d; single thread; interpolation type is set to linear. For Voreen, we enabled floating point buffers in the pipeline. The Voreen version under verification does not support parallel projection.

The errors are computed using the L_∞ norm and are given by the maximum distance between two images, defined as $E_i = \max_{x,y} |I_i(x, y) - I_{i+1}(x, y)|$, where $I_i(x, y)$ is the pixel with center in (x, y) of the image I_i rendered with the implementation under verification. If a solution F is available, $E_i = \max_{x,y} |I_i(x, y) - F(x, y)|$.

In the following sections we report the results of applying the verification framework with known and unknown solutions to three volume rendering implementations. Table 5.2 indicates where one can find the verification results based on the type of solution (known or unknown), the convergence parameters, and also the projection step used.

5.5.3 Observed behavior

The results of our verification procedure are summarized in Figure 5.7. We tested both VTK and Voreen and found unexpected behaviors. We emphasize that this *does not* immediately translate into a code mistake but only that a deeper analysis is needed. To find the reason for the unexpected behavior we analyzed the source code of the given systems. We expect linear convergence when step size or pixel size are refined ($k = 1$) and zeroth-order convergence when dataset refinement is used ($k = 0$).

FP The results obtained for the FP module (blue curves in Figures 5.7(a), (b), and (c)) were different from expected for all tests. The 15-bit fixed-point precision could, to some extent, justify this behavior. Still, we only expected this influence to have a negative

Table 5.2. Each cell indicates where the results for each type of test can be found in the paper.

		Known sol.	**Unknown sol.**	
		Parallel	*Parallel*	*Perspective*
FP	*step size*	Fig. 5.9	Fig. 5.7(a)	Fig. 5.7(a)
	pixel size	Fig. 5.7(b)	Fig. 5.9	Fig. 5.7(b)
	dataset size	Fig. 5.9	Fig. 5.7(c)	Fig. 5.7(c)
RCM	*step size*	Table 5.3	Fig. 5.7(d)	Fig. 5.7(d)
	pixel size	Fig. 5.7(e)	Fig. 5.10(b)	Fig. 5.7(e)
	dataset size	Fig. 5.10(a)	Fig. 5.7(f)	Fig. 5.7(f)
Voreen	*step size*			Fig. 5.7(g)
	pixel size	N/A	N/A	Fig. 5.7(h)
	dataset size			Fig. 5.7(i)

effect after a certain threshold for step size. The perspective projection curve shown in Figure 5.7(a), for instance, has zeroth-order convergence when using step size refinement and perspective projection. We expect convergence at least for large values of d because when d is too small, the errors due to 15-bit fixed-point precision will dominate. After investigating the reason for this deviation we found that depending on the pixel position, some rays might cross only half of the volume instead of the full volume. In other words, instead of sampling the ray in n locations, for some pixels the ray was only sampled $\frac{n}{2}$ times. This is a combination of several factors which includes domain size, step size, and ray direction. Details can be found in the supplementary material.

Using our synthetic dataset, we observed a '+' pattern shown in Figure 5.9 (left). The darker regions are precisely the pixels where the ray does not cover the whole domain. Artifacts may also be seen in standard datasets such as the Carp shown in Figure 5.8. The orange curves in Figures 5.7(a), (b), and (c) show the convergence results after modifying VTK's source. Notice that for step size refinement using perspective projection, the convergence curve changed from 0.02 to 0.92 for the first seven samples. For the eigth and ninth samples the error slightly increases. A similar phenomenom occur in the parallel convergence curve. The curve starts to diverge in the high-resolution regime (parallel and perspective projection plot). This might be due to the limit of 15-bit fixed point arithmetic. Although the pixel size refinement convergence for perspective projection substantially improved (from 0.01 to 0.94), the convergence curves for parallel projection remained similar. At this point, we do not know the source of the discrepancy: the testing suite, the theory or the implementation under verification.

RCM The RCM module (blue curves in Figures 5.7(d), (e), and (f)) produces nearly linearly converging sequences when refining the step size or pixel size. However, dataset refinement with either perspective or parallel projection fails to present the expected zeroth-order convergence. Analyzing the source code, we found the discrepancy to be due to the number of steps taken when marching inside the volume. For instance, suppose that the step size is set in such a way that 200 steps are required to traverse the volume. Instead of 200 steps, the RCM module used from 195 to 199 steps, depending on some conditions.

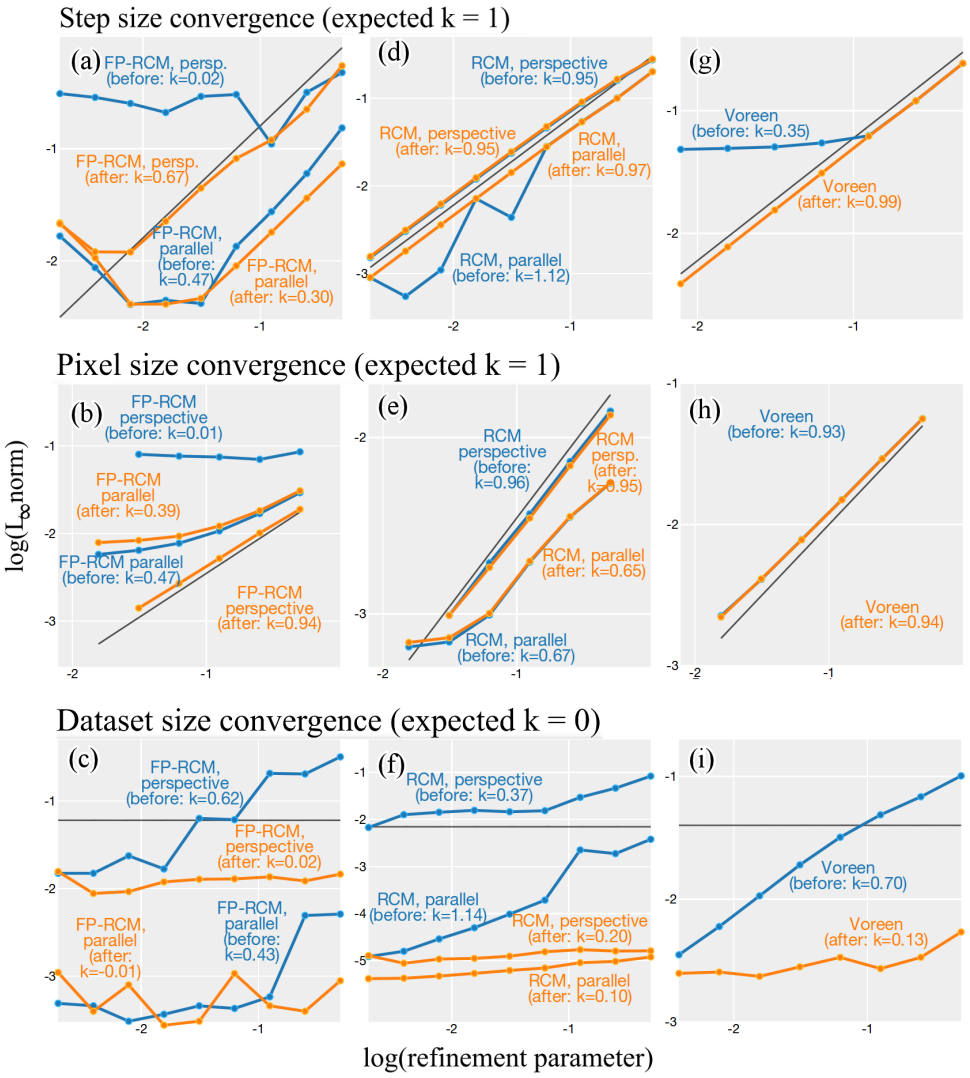


Figure 5.7. Each plot shows the convergence experiments for one particular implementation and one particular type of convergence. The behavior before any changes to the source code are shown in blue. The results of the changes are shown by the orange lines. The black line indicates the expected slope from the theoretical analysis. Notice the black lines indicate only the expected *slope* of the results. Any line parallel to the black indicator line has the same slope and is equally acceptable. The line slope value is denoted by k.

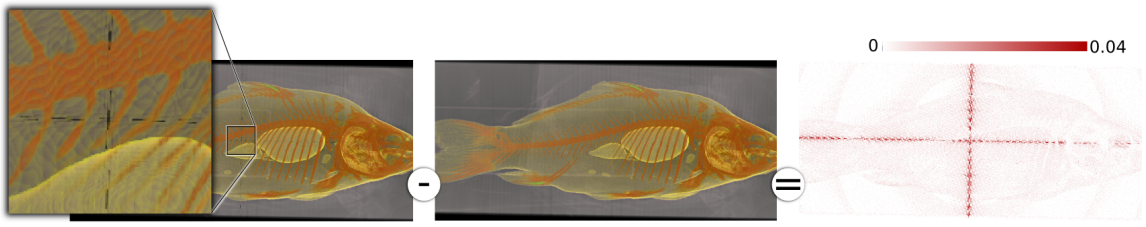


Figure 5.8. A CT scan of a carp, rendered with VTK 5.6.1 and Fixed-Point Raycast Mapper (FP). On the left, we see the artifacts (dark lines) that prevented FP convergence. On the middle, we see the results after fixing the issues which prevented convergence. The artifacts are no longer visible. On the right we see the difference image.

The consequence of this deviation is shown in Figure 5.10.

The orange curves in Figures 5.7(d), (e), and (f) show the convergence results for the RCM module after fixing the issue that prevented code convergence. It consists of changing the epsilon values used during the computation of the number of steps. Notice that the behavior is close to the expected one and the errors are very small (10^{-5}). The convergence curve using pixel size refinement is close to linear for large pixel size but seems to be converging to some positive value. This might be due to other sources of error which become dominant after sufficient refinement.

Voreen Our first ray refinement tests did not result in linear convergence for Voreen (blue line in Figure 5.7(g)) due to the early ray termination (ERT). By simply adapting the ERT threshold, we were able to obtain the expected convergence for ray refinement (orange line in the Figure 5.7(g)).

As can be seen in the Figure 5.7(i), the blue curve indicates that increasing the resolution of the dataset decreases the error. We remind the reader that using our upsampled data, as described in Section 5.3.2, rendering the same scalar field represented by a different number of voxels should not affect the result. For Voreen, the unexpected behavior was caused by sampling at incorrect texture locations. More specifically, internally, Voreen assumed that the texture data is node centered when, in fact, OpenGL uses grid centered data. In this case, both the volume and transfer function values were affected. In OpenGL, the texture coordinates of a texture of resolution R^m lie in the domain $[0,1]^m$, where m is the texture dimension. Since the data values are grid centered, this means that the outer most data values are located at $[\frac{1}{2R}, 1-\frac{1}{2R}]$ with the settings used in Voreen. We will refer to the domain in which the data values lie, as the data domain. For volume rendering, the integration of a ray should be done over the data domain, but for Voreen, the entry and

exit points of the rays went outside of that domain which caused the unexpected behavior. To obtain the expected constant convergence we apply the following transformation to the input texture coordinate p (see orange line in Figure 5.7(i)):

$$p' = \frac{1}{2R} + p\left(1 - \frac{1}{R}\right), p \in [0\ 1]. \tag{5.52}$$

Equation (5.52) scales and translates the texture coordinate to be in the domain $[\frac{1}{2R}, 1 - \frac{1}{2R}]^m$, where the data values lie. The effect of transforming the input coordinate for a real world example can be seen in Figure 5.1. We provide an explanation for why this does not affect ray entry and exit point sampling, and also discuss the implications of different boundary values in the supplementary material. Although the scaling of texture coordinates has been addressed for multiresolution volumes [93], to our knowledge, it has not been applied to the sampling of transfer functions [39, 80, 140]. There are other solutions for recovering the expected convergence rate, which include changing the way we refine our sampling grid to match OpenGL grid centered data. However, we have chosen this solution for several reasons. First, it matches Voreen's initial assumption on node-centered data; it does not require special treatment at the border of the domain; and due to its simplicity, it is easy to implement. We have contacted Voreen developers and the issue found was indeed identified as a bug and the proposed solution will be adopted into Voreen's next release.

No unexpected behavior could be detected for pixel size convergence as shown in Figure 5.7(h), neither before nor after changing the texture coordinate sampling. Both curves lie near the expected behavior (0.93 and 0.94).

5.6 Discussion

The convergence analysis presented in the previous section helped us to identify unexpected behavior in two stable and widely used frameworks. Unexpected behavior *is not* indicative of an implementation bug but rather a warning about potential problems. For instance, some valid design decisions might affect the convergence results. Consider the widely used ERT acceleration technique. Depending on the thresholds involved, the convergence results might deviate from the ideal, and the expected curve is recovered once this feature is turned off. In this sense, the verification tool can help the developer to identify portions of the code that introduce numerical errors and quantify their effect on the final image. The issue with the RCM module is another example. The dataset size convergence curve was unexpectedly linear because of a small variation in the number of steps. While this particular issue might not be harmful, we were able to learn and reason

about its consequences *after* the verification process was done. Furthermore, "minor" bugs and even design decisions cannot be ignored as they can mask more complex mistakes. Therefore, one will be more confident after the design decisions that affect convergence are "turned off" and the expected convergence is recovered. The FP module, on the other hand, significantly deviates from the ideal number of steps required to march inside the volume. Although we could force VTK to march the expected number of steps, we are still investigating possible solutions to and consequences of this issue. To promote an unexpected behavior to a bug, we need interaction with the developers of the code to confirm the code mistake, which was the case with Voreen. One should be aware of the discussed issues when implementing a volume rendering algorithm as their consequences are often not discussed in the literature [39].

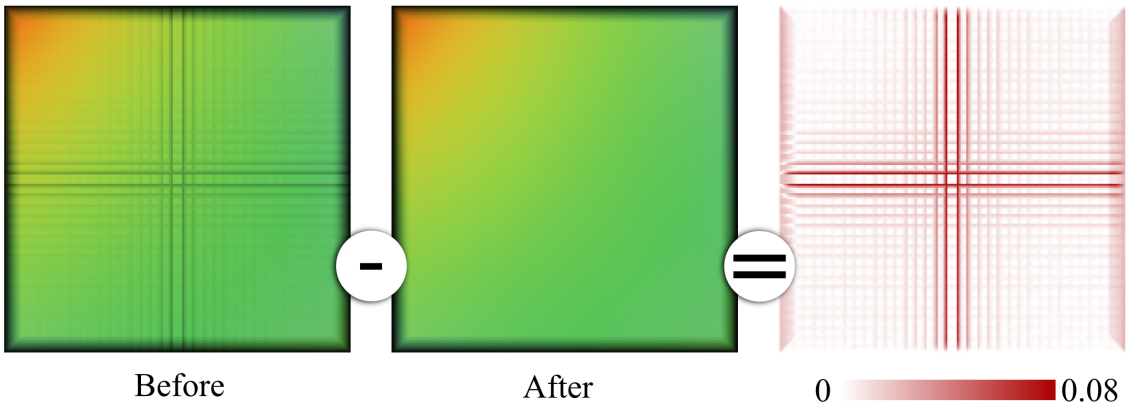
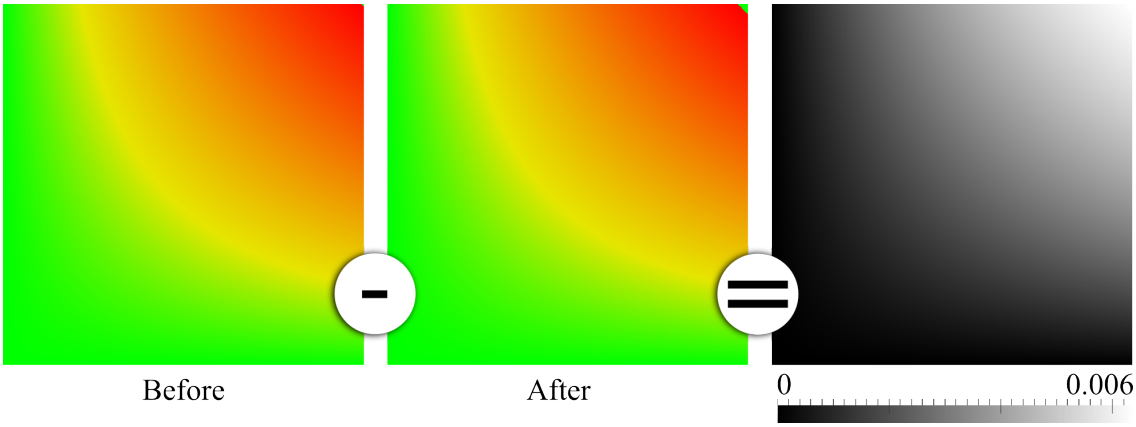
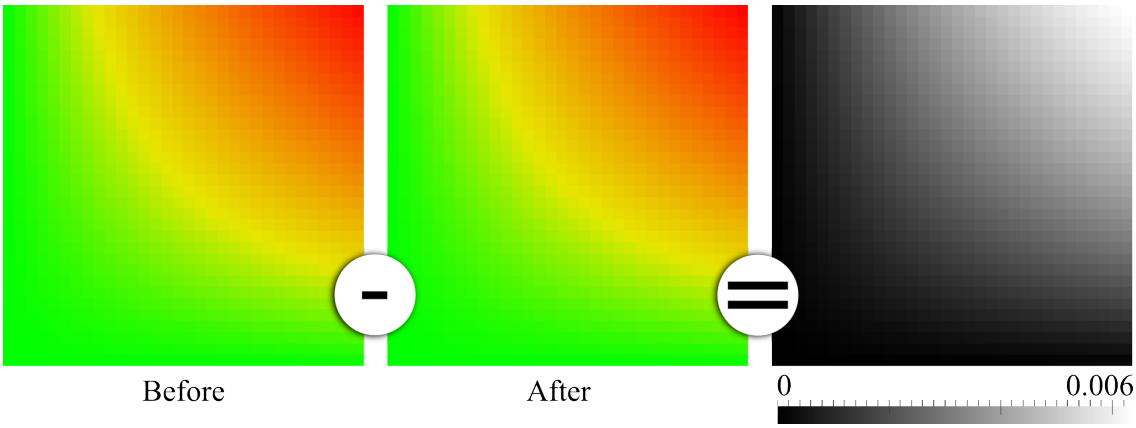


Figure 5.9. The figure shows images rendered using VTK 5.6.1. In our experiments, the '+' pattern became more evident in two cases: when coarse datasets are used; and/or high number of sampling points along the ray are used. Darker pixels belong to regions where the ray traverses only half of the volume, preventing convergence. The image on the middle shows the result using our modified version of VTK. The convergence curve improved significantly. Note that this effect only occurs when perspective projection is used. For orthogonal projection, the problem is not noticeable. Step size convergence: Expected $k = 1$. Before $k = 1.2$. After $k = 1.2$. Dataset size convergence: Expected $k = 0$. Before $k = -0.02$. After $k = -0.02$. Pixel size convergence: Expected $k = 1$. Before $k = 0.84$. After $k = 0.85$. For the convergence analysis, we used a scalar field given by $S(x, y, z) = xyz$, $D = 1$, transfer function $\tau(s) = s$ in the domain $[0, 1]^3$, and solution for the VRI given by $I(x, y) = 1 - \exp(-xy/2)$, which means the integration is along z (from zero to one).



(a) Dataset refinement. Exp.: $k = 0$. Before: $k = 0.75$. After: $k = -0.18$



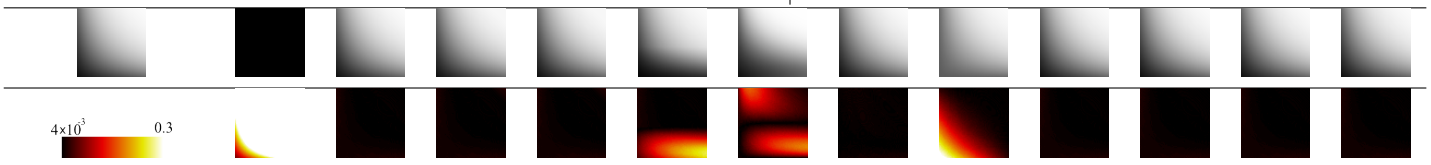
(b) Pixel size refinement. Exp.: $k = 1$. Before: $k = 0.37$. After: $k = 1.23$

Figure 5.10. The two figures show images rendered before and after fixing an issue with the number of ray samples in the RCM module. This change was motivated by a mismatch in the dataset convergence test. Although the images are indistinguishable to the human eye, the errors (computed as the difference between images, shown on the right) are large enough to change the expected convergence rate. For both images, we applied our verification procedures on a grid with a scalar field given by $S(x, y, z) = xyz$ and transfer function $\tau(s) = s$ in the domain $[0, 1]^3$. Hence, the solution for the VRI is $I(x, y) = 1 - \exp(-xy/2)$. (a) uses dataset refinement while (b) uses pixel size refinement.

Table 5.3. This table shows the sensitivity of the convergence verification for different scenarios in a volume renderer. We applied our step size verification using a manufactured solution with a scalar field given by $S(x, y, z) = xyz + xy + xz + yz + x + y + z + 1$ and transfer function $\tau(s)$ varying linearly from zero to one for $s \in [0, \max(S(x, y, z))]$. On the right, we show what part of the volume rendering algorithm was affected by the issue. In the bottom, the first row shows the rendered images for each of the issues. The second row shows the error difference between the exact and rendered solutions. See Section 5.6.1 for an explanation of the undetected issues.

#	Issue	Detected	Observed behavior ($k =$)
1	Incorrect opacity accumulation	Yes	0.0
2	Incorrect ray increment	Yes	0.0
3	Small changes to early ray termination	Yes	0.1
4	Piecewise constant τ	Yes	0.0
5	Incorrect matrix-point multiplication	Yes	0.0
6	Incorrect evaluation of trilinear interpolant	Yes	0.0
7	Uninitialized pixel center offset	Yes	0.0
8	Incorrect coefficients computation 1	Yes	0.0
9	Incorrect coefficients computation 2	No	1.0
10	Incorrect color lookup	No	1.0
11	Incorrect matrix-viewpoint multiplication	No	1.0
12	Incorrect loop range	No	0.95

```
Volume Rendering
    for each pixel
        do Find pixel center (#7)
           Transform rays to voxels space (#5, #11)
           for each step along the ray (#12)
               do Compute interpolant coefficients (#8, #9)
                  Interpolate scalar values (#6)
                  Retrieve color and opacity (#4, #10)
                  Compositing (#1)
                  Increment sample position (#2)
                  Check for early ray termination (#3)
```



Exact solution 1 2 3 4 5 6 7 8 9 10 11 12

5.6.1 Test sensitivity

A verification technique ideally should be sensitive to any deviation from the correct implementation. Unfortunately, in practice, verification has limited scope, and we gain confidence if it helps us understand the code behavior, test sensitivity, and reveal bugs. There are several ways to attain this goal: Yang *et al.* applied model checking for filesystem verification and reported *unknown* bugs [187]; Howden [61] evaluated the efficacy of dynamic and static testing for the detection of *known* real bugs of a mathematical library; Knupp and Salari [76], on the other hand, used the order-of-accuracy verification procedure to uncover *known* manufactured bugs in a proof-of-concept code. In software engineering, the process of evaluating a testing suite by injecting defects into a program is known as *mutation testing* [138].

We already presented the results of applying our verification framework to two libraries and with our experiments we confirm the previously reported sensitivity of convergence analysis [142]. We went further to explore other scenarios in volume rendering that may affect the convergence curve. Thus, in the spirit of mutation testing, we created new versions of VTK which contain known issues. Table 5.3 shows the results of some of the performed tests. In our experiments, we observed that some issues did not affect the observed behavior. The reason for this is that an incomplete set of tests [76] was performed, as shown with test #10 in Table 5.3. In that case a bug in the G and B color lookup went unnoticed because our framework only used the R channel. Once the verification framework includes all three channels, the convergence drops to 0, revealing an unexpected behavior. For bug #9, we swapped two of the polynomial coefficients but they were equal for the scalar field used and thus it was not detected. After changing the scalar field to $s(x, y, z) = 1xyz+2xy+3xz+\cdots$ the convergence curve no longer matches the expected one and thus the bug is detected. Bug #11 was introduced in a matrix-vector multiplication routine which turned out to be dead code. However, for bug #12, the loop range was slightly incorrect and it was not detected, even after additional changes to the verification framework.

Aside from the defects injected into VTK, the following is a list of details known to affect the convergence curve: *ERT*, as explained before; *opacity correction*, when using the analytical solution of the volume rendering integral; *hardcoded tolerance constants*, the famous "epsilons"; *off-by-one* indexing problems (sometimes VTK does not render pixels in the first or last column of an image); *improper volume sampling* (cell centered versus grid centered scalar fields); *high-frequency transfer functions*; *high-frequency scalar fields*; *incorrect texture coordinate mapping*, as reported with Voreen; *inconsistent number of steps*

through the volume, as reported with FP and RCM; etc. From all the observed situations where the step/dataset/pixel size convergence was affected, many of these are deliberate design decisions, minor mistakes during the verification procedure or minor problems with the implementation itself which can be easily fixed. Note that those issues were not all inside the ray integration routine itself, but in a variety of locations, spanning from pre-processing steps to OpenGL texture sampling of data. Our verification procedure was sensitive enough to detect all these situations.

We stress that our goal is to provide a methodology for verification of volume rendering implementations. Specifically, our goal is neither to compare different volume rendering implementations nor to entirely explain the behavior of the examined implementations. The users of the proposed framework are developers and practitioners who want or need to identify potential problems.

5.6.2 Other volume rendering techniques

While we focus on ray casting, our approach can be extended to other techniques. Because the core of our method is a discretization of the VRI, the only requirement is to formulate the volume rendering algorithm as a numerical approximation to the true integral. Splatting [181], for instance, uses a reconstruction kernel before accumulating the contributions of voxels into the final image. This approach is substantially different from ray casting in the way it approximates the VRI, and so the asymptotic errors involved will have to account for errors in both accumulation and filter reconstruction [110].

Algorithmic improvements for volume rendering may require a more careful approach. For example, pre-integration computes the results of the integral with high precision over sample intervals and stores them into a look-up table. This increases efficiency and quality, since fewer steps are typically needed [40]. How the table approximates the integral will affect the convergence rate: if there is an analytical solution then no error is associated with d intervals; otherwise, a numerical approximation scheme might be used which means the error will depend on $d' = d/m$, where m is the number of sample points used in that interval and the integration method used. For example, if a linear approximation is used for the VRI during ray integration (instead of a standard sum of rectangles, as done above), the final approximation should have second order accuracy.

5.6.3 Manufactured solutions

In the interest of brevity, verification via pixel size and the results presented in Table 5.3 were generated from an analytical solution for the volume rendering integral. Notice,

still, that the use of an analytical solution for verification is known as the Method of Manufactured Solutions [3] and can be a more rigorous procedure than convergence analysis alone [142]. In this way, we can verify that the results generated by an implementation is converging with the right speed to the *correct solution*. The disadvantage lies in the difficulty of designing manufactured solutions which are simultaneously simple (so that we can write the theoretical convergence analysis down) and sensitive (so that the experiment analysis catches bugs).

5.7 Limitations

Both the discretization and verification procedures have limitations. In the discretization of the VRI equation, we assume that the solution $I(x, y)$ is smooth. Moreover, we assume that high-order terms are negligible. This assumption implies that we can safely discard all high-order terms when deriving the errors. In addition, the verification is done in a controlled fashion to avoid other error sources, as shown in Figure 5.7(a). Additional asymptotic analysis is necessary for each new error source. Also, I must be defined everywhere in the image plane. For instance, this condition is violated if we change the camera position and orientation. One needs to account for these transformation in $\mathbf{x}(\lambda)$, an extra complication in the generation of analytical solutions.

The verification process has the same limitations previously described but it also has practical limitations. For instance, one may be able to observe that the convergence rate may not be the expected one for low sampling rates. However, this is not due to the random scalar field generated (which is a trilinear function and thus can be represented exactly with the trilinear interpolant) but high-frequency details in τ or C. This may lead to a violation of the Nyquist rate. Because the process is iterative, for a correctly implemented code, the expected convergence must be recovered once the resolution is fine enough. Another limitation is related to the number of rays used per pixel. Many implementations can shoot several rays per pixel, although this work assumes that only one ray is used. Also, because the verification procedure considers the code as a blackbox, it does not provide clues on the reasons for the unexpected behavior.

The scope of the mistakes that can be found by the verification procedure is not clearly defined. All we can say is that it can find bugs that actively affects the convergence of the method [76]. A common example of bugs that cannot be found by this type of procedure is bugs that affect the *performance*: the code is slower due to the mistake but the convergence is still the same [139]. The results shown in Table 5.3 is a first attempt to understand the

scope of problems that can be fixed by the verification procedure.

Currently, our verification procedure is focused on the solution for the VRI without shading and other improvements on the final image quality. Hence, if one wants to use our verification procedure in an implementation that supports, for instance, shading, the feature will need to be deactivated. Lastly, for the case of dataset refinement, we assume that the underlying scalar field is defined by a piecewise-trilinear function.

5.8 Conclusion and Future Work

In this paper, we present verification techniques for volume rendering based on the use of convergence analysis. Using these techniques, we successfully found discrepancies in the behavior of the volume rendering algorithms of two widely-used visualization packages. We note that we do not see our techniques as a replacement for the currently used direct visual inspection or expert evaluations, but instead as a way to complement those approaches, and lead to a more comprehensive way to evaluate visualization software. By providing attractive quantitative alternatives, we hope to help make evaluation of visualization software both easier and more effective, and also contribute to a higher level of user trust in visual data analysis. We believe the use of verification techniques will be of increasing importance as the field of visualization matures and visualization methods are used in a wide range of commercial and societal areas of highest importance.

There is ample opportunity for future work. Extending our approach to deal with volume shading and level-of-detail techniques would be interesting and relevant research as these are widely used in practice. Another important problem would be to explore the verification of unstructured volume rendering techniques. Lastly, there is room for improving the approximation error for the three presented refinements. In addition, a new way for comparing the convergence curves that allows one to gain insight on the correctness of the implementation under verification is another welcomed step.

CHAPTER 6

RELIABLE VISUALIZATIONS

Flow visualization has been around in some form for as long as people have studied flows. In some cases, visualization was done explicitly – that is, with the expressed purpose of the viewer to highlight some feature of the flow. In other cases, it was done tacitly, as when a child looks out the window of an airplane to see the slip-stream over the wing generated upon take-off. Visualization has many roles, spanning from art to science. In this work, we focused on visualization techniques used for the scientific exploration and explanation of flow phenomena. In particular, we are interested in how two communities – the AIAA community and the Visualization community – consider flow visualization. To accomplish this task, we have used the *AIAA Journal* and the *IEEE Transactions on Visualization and Computer Graphics (TVCG)* as "representative" publication venues of the two communities, and have explored the papers published therein to try to glean how each community approaches visualization of flow, how they might differ from each other and how the two communities might complement each other.

This work is organized as follows. In Section 6.1 we provide a review of the state-of-the-art in flow visualization, both from the perspective of the Visualization and well as the AIAA communities. Tools such as Tecplot[2] and Paraview[159] have implemented many standard flow visualization techniques such as LIC (line integral convolution), streamlines, stream ribbons, and more. As we will show, our review encompasses much of the current practices in flow visualization and also provide pointers to new developments. In the next two sections, we focus our attention on research advances made within the Visualization community that we think will, in time, have impact on flow visualization and on other application domains that use visualization as a means of both scientific exploration and explanation. In Section 6.2 we show how perception and user studies may impact flow visualization, and in particular, we focus on issues related to color maps. In Section 6.3, we then provide discussions on the current Visualization community research trends in Visualization Verification and Uncertainty Quantification. We have chosen these topics

because they are all related to flow visualization. In Section 6.4, we speculate on some of the opportunities for collaboration and more effective communication between the two communities, and we conclude in Section 6.5.

6.1 Review of Flow Visualization Techniques

Vector field visualization is an important and vibrant subfield of both the Visualization and AIAA communities. The techniques developed for vector field visualization extend beyond these communities to fields such as medical imaging, meteorology, the automotive industry, and others. In the past two decades, visualization experts and practitioners have seen the development and improvement of many vector field visualization techniques. The contributions are numerous: the ability of handling different grid types (structured, unstructured, curvilinear, etc), high dimension data (2D, 2.5D, and 3D), time-dependent flow, seeding and placement of geometric primitives, improved performance, perception, rendering, among others. In this section, we review some of the developments inside the Visualization community and compare with current practices inside the AIAA community.

6.1.1 Preliminaries

Although the concept of flow visualization is well defined in both communities, we start by clarifying what is meant by flow visualization in this section. The difference between *computational flow visualization* and *flow visualization* is that the latter focus on visualization of flow behavior using experimental data (*e.g.*, flow in a wind tunnel), whereas the former visualizes flow from simulated or computed data. Some computational visualization techniques are inspired by techniques used in flow visualization, such as dye advection. Since the subject of this section only addresses computational flow visualization, we will refer to that topic simply as flow visualization.

For thoroughness, we also define some commonly used mathematical/physical terms used within the flow visualization literature. A *streamline* is the path traced by a massless particle in a steady flow. Streamlines are sometimes referred to as "instantaneous particle trace". A *streakline* is the path traced by massless particles seeded at the same position but at different times in a unsteady flow. *Stream surfaces* and *streak surfaces* are the 2-manifold analog of streamlines and streakline, where the seeding primitive is a curve instead of a point.

Table 6.1. Advances in flow visualization. This table is *not* meant to be comprehensive.

Class	Subclass	Technique	Reference
Direct	*Arrows*	Standard	Klasshen and Harrington[74]
		Hybrid	Color-coding and arrows[73]
		3D	Arrows in 3D space, 2-manifolds embedded in 3D[129]
		Enhancements	Large data[129], resampling[85]
	Color coding	Standard	Color maps, volume rendering[38]
Geometry	*Curve*	Streamline	Turk and Banks[171]
		Seeding	User-assisted[64], automatic[105, 92], and hierarchical[65]
		3D	2-manifolds embedded in 3D[158]
		Rendering	Illuminated[100], streamtubes and streamribbon[172]
		Unsteady	Wiebel and Scheuermann[182]
	Surface	Stream surface	Hultquist[62]
		Enhancements	Seeding and placement[128], accuracy[47]
		Unsteady	Schafhitzel *et al.*[146]
Texture	*LIC*	Standard	Cabral and Leedom[14]
		Performance	Improved algorithm, parallelism, real-time, GPU[91]
		3D	3D and 2-manifolds embedded in 3D[125]
		Rendering	Flow orientation cues, local velocity magnitude
		Unsteady	Li *et al.*[91]
	Spot Noise	Standard	van Wijk[176]
		Enhanced	It deals with highly curved/high velocity vector fields.[29]
		Performance	Parallel implementation.[87]
Feature	*VFT**	Standard	First-/High-order critical point tracking[57, 28, 148]
		Compression	Theisel *et al.*[164]
		Simplification	Weinkauf *et al.*[180]
		Streakline	Weinkauf and Theisel *et al.*[179]
	*STD***	Pathline	Theisel *et al.*[166]
	*LM****	FLTE	Haller[54], Garth *et al.*[46]

* Vector Field Topology ** Space-Time Domain *** Lagrangian Method

6.1.2 Classes of techniques

Flow visualization techniques can be classified as direct, geometric, texture-, and feature-based. Table 6.1 provides an overview of the classification and a subset of the available techniques within each class. The table provides a hierarchy of the flow visualization tools available. The *Subclass* column provides the main component of a given visualization techniques that can be found within the *Technique* column. One can find reference to extra material within the *Reference* column. For more details about the articles shown in Table 6.1 and others, we refer the interested reader to the excellent surveys by Hauser *et al.*[56] and Peng and Laramee[130] for an overview of the flow visualization field, Edmunds *et al.*[36] and McLoughlin *et al.* [104] for geometric flow visualization, Laramee *et al.*[84, 83] for texture-based flow visualization, and Pobitzer *et al.*[132] for feature-based flow visualization. Next, we briefly go over each of the classes (see Figure 6.1).

6.1.2.0.1 Direct visualization Direct visualization techniques provide an intuitive and straightforward way of visualizing vector fields. In this approach, primitives of interest – such as arrows, glyphs, or lines – are placed at (often regularly-spaced) seed points. The primitives are then oriented according to the vector field. Optionally, the vector magnitude can be mapped to the primitives via scaling. Other flow properties, such as pressure and vorticity, can also be mapped using color maps. In the 3D case, volume rendering[38] is the natural choice for mapping flow properties into color and transparency. Although direct visualization provides an easy first approximation of the vector field, the visual complexity and occlusion may impair the interpretation of the results, especially in 3D datasets.

6.1.2.0.2 Geometric visualization In geometric visualization, curves and surfaces are used for summarizing flow behavior at particular seed points. Geometry-based approaches requires a more intensive processing of the data before the visualization than direct approaches. The main idea behind integration-based geometric flow visualization is to trace particles or curves through the vector field. By tracing particles (or respectively curves) one builds a 1-manifold (or respectively a 2-manifold) that can later be visualized. Geometric visualization techniques have a two steps: first, geometry computation; and

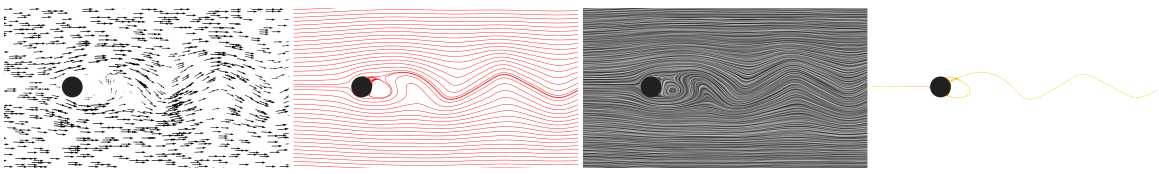


Figure 6.1. Examples of flow visualization using direct, geometry, texture-, and feature-based techniques, respectively.

secondly, rendering. Often, the rendering step is straightforward – *e.g.* rendering a polyline – in which case the algorithm collapses into one step. Streamlines are one of the most well-known representative visualization tools within this class. Although flow visualization using both curves and surface dates back over two decades, in recent years there has been constant research on the topic[36]. For curves, the main contributions of the past decade are related to rendering, seeding and placement of curves. Edmunds *et al.*[36] classifies the surface-based flow visualization into surface construction and rendering. Methods for surface construction are based on integral surface, implicit and topological construction. This is an area of intense research in the past few years. The authors present a variety of algorithm for both steady and time-dependent surfaces. Surface rendering methods involve the use of several techniques for improving the quality of the visualization of the flow over a surface of interest. Surface-based techniques can take advantages of direct or texture-based methods by including static/animated arrows over stream surfaces, shading for the evaluation of the shape of surfaces, placing streamlines over 3D surfaces, employing line-integral convolution (LIC) techniques, and/or non-photorealistic rendering techniques.

6.1.2.0.3 Feature-based visualization In feature-based flow visualization, the input vector field is segmented according to features of interest. As an example, consider a segmentation using classical vector field topology in 2D[57] (see also the right image in Figure 6.1). Let us assume that the features of interest are first order critical points, namely, focus source, focus sink, node source, node sink, and saddles. A segmentation is performed by building a topological skeleton through the computation of the vector field's separatrices. The final result provides a cleaner representation of the flow behavior in terms of the aforementioned features. The intensive processing of extracting features before visualization brings many advantages to the practitioner. First, feature-based techniques are valuable for visualization purposes: feature extraction provides an excellent level of abstraction of the data by removing undesired features and focusing the viewer on the important regions of the dataset. In addition, it can be used for vector field compressing, topological simplification, and even for building custom vector fields[165]. Topology-based approaches for feature-based visualization is not the only methodology available. In Lagrangian methods, the trajectories of particles are used to describe and segment the fluid flow. In particular, FLTE[54] methods have gained prominence as a research area within the last decade. One advantage of Lagrangian methods over traditional vector field topology is that they can naturally deal with unsteady flow[132]. Space-time domain techniques are another example of feature-based visualization. In this approach, in order to deal with

the problems involved in unsteady flows, the problem of 2D and 3D flow visualization is moved to higher dimensions. As an example, time-dependent domains are merged into a single dataset where traditional techniques used for steady vector fields can be employed. A comprehensive survey on the topic can be found in the state-of-the-art report by Pobitzer *et al.*[132].

6.1.2.0.4 Texture-based visualization In texture-based flow visualization, the user replaces geometrical information with 2D texture mapped over surfaces. Line integral convolution (LIC) is a well-known (within the visualization community, at least) representative of the class. Texture-based techniques generate what is considered a dense visualization, *i.e.*, it covers the entire domain of interest, and it does not have to deal with the problem of finding appropriate seeding spots for streamlines. Texture-based techniques can be applied along with geometric or feature-based visualization; for instance, it can be used to render flow on 2-manifolds embedded in 3D spaces, or providing an overview of the flow behavior along with topological skeletons. The main issue with texture-based visualizations is the high computational cost associated with it. Nevertheless, the advances in both computer hardware and algorithms have granted to users the ability to handle large data sets and unstructured grid at interactive rates[36, 83].

6.1.3 Means to an end

In his position paper "On the death of visualization"[95], Lorensen argues for the need to bring visualization researchers closer to experts and practitioners. We have run a simple experiment in order to attempt to ascertain "the distance" between the Visualization and AIAA communities. We evaluated 78 articles published within the *AIAA Journal* over the period of Jan/2010-Oct/2012 containing at least one flow visualization image. Then, we simply counted the number of papers that contained at least one occurrences of the techniques shown in Table 6.1. We did not include the 2D color mapping and 2D isocontour visualizations as they appear quite often. Since multiple visualization techniques can be used in a single article, the percentages shown below are just the fraction of publications containing at least one particular type of visualization. Particle tracing using integration-based geometric visualization techniques for 2D vector fields is the most commonly used technique (42%), followed by 3D isocontouring (35%), 2D and 3D arrows and glyphs (33%), and 3D particle tracing (19%). Excluding isocontouring (which is mainly used for depicting scalar, instead of vector, data), 61% of the articles used at least one geometric approach to flow visualization, whereas 33% used a direct approach. Finally, 73% of the papers contained at least one visualization for 2D domains, whereas this number is 56% for 3D

domains. The latter number drops to 22% if one considers only techniques for visualization of vector field data (*i.e.*, excluding 3D isocontouring).

Although the data is limited to a short window of time, it raises a few interesting points. With the exception of a handful of papers, most of the flow visualization appears to be using the standard form of the traditional visualization technique. As an example, consider some the papers that use streamlines for visualizing 3D flow. It may be the case that a subset of these paper can benefit from using stream ribbons[172], which simultaneously encode the streamlines path and local flow vorticity, or from stream tubes[172], which simultaneously encode the streamlines path and local cross flow divergence. Both stream ribbons and stream tubes are well-known, and commonly used visualization packages such as Paraview or Tecplot have them available within their tool options. Secondly, the preference for the two visualization techniques (direct and curve-based geometric visualization) shown in past three years is perhaps due to their simplicity and availability. The underrepresented methods in the same period of time are texture-, feature-, and surface-based flow visualization. Third, one could argue that the visualized datasets were "simple", and thus standard techniques worked well. Even though this may be the case for some datasets, some vector fields, especially in 3D, suffered from traditional problem of curves and arrows: cluttering, irregularly spaced streamlines, poor seeding, lack of depth cues, *etc.* These problems can make the detection of some flow features such as vortex more difficult. Direct visualization for 2D vector fields using glyphs can be improved by using, for instance, a resampling technique, such as shown in Laramee[85], where the author introduce a user-driven approach for reducing visual clutter via resampling. Another way is to segment the flow using features of interest, *e.g.* critical points. Possible reasons for *not* using alternative techniques include that the technique might not be easily available, the technique might not improve the quality of the visualization, users not aware of their existence or find them difficult to use, or the AIAA community requires a different class of techniques, among other. Both communities would benefit from knowing the reasons for using one technique over another. The visualization community has, throughout the years, defined a set of priorities based on an interaction with researchers from different fields and their own experience. Some recurrent themes that are the focus of research are: a more comprehensive theory and techniques for dealing with unsteady 3D flows; improved rendering (for instance, by using techniques inspired in handcrafted illustrations[13]); handling of large data sets; and others. Together, the AIAA and Visualization communities should be able to define a set of priorities for their research agendas in order to address the concerns and issues raised.

6.2 Perception and Evaluation

An important aspect of the visualization research consists of the building of new visualization techniques and tools. Ideally, new techniques should be able improve the user cognitive process[169], for instance, by allowing the visualization of data that has never been visualized before, or increasing ones ability to interact with, understand, and explore data. As visualization techniques are developed and improved, a question is raised: how can we compare and understand the differences between visualization techniques? The answer to this question leads us to a second important research topic: the need for rigorous evaluation of the strengths and weaknesses of visualization techniques. By "strength" and "weakness" we mean not only the evaluation of techniques according to traditional (computer science) metrics such as performance, memory footprint, ability to handle large datasets, *etc.*, but also in terms of the errors introduced through visualization, property of these errors, user perception, among others. In particular, questions involving perception and cognition are related to the *user*. In this section, we review two topics of interest for flow visualization from the point of view of perception and evaluation: the use of color maps for visualization of scalar properties and the representation of steady 2D vector fields, respectively.

6.2.1 Perception & color maps

The mapping between data and colors is ubiquitous and essential across the sciences. In the scientific pipeline, color maps are often used to study, explain, explore, and ultimately help experts to gain insight about a phenomenon of interest. Alas, color maps are not all equal, and depending on the choices made one can accelerate or impair scientific inquiry. Since they are just means-to-an-end, their impact on the underlying data should be as minimal as possible. In a myriad of choices, one color map has been shown to be a bad choice for virtually any type of visualization: the well-known and widely-used *rainbow* color map.[10, 97, 156].

The rainbow color map is built by varying hue in order to cover the whole spectrum of visible light, from red to purple or vice versa. In practice, many visualization tools use colors varying from red to blue because red and purple are very similar. It is the default map in several visualization / simulation software packages, such as Matlab®. Here we review three issues known to hinder visualizations, namely, lack of ordering, iso-luminance, and introduction of artifacts. Figure 6.2 shows examples for each of these issues. The first issue is due to the lack of a *natural sorting order*. Even though the rainbow color map is ordered from shorter to longer wavelength of light, users do not easily perceive it as such, which makes quantitative analysis more difficult[10]. In addition, the rainbow

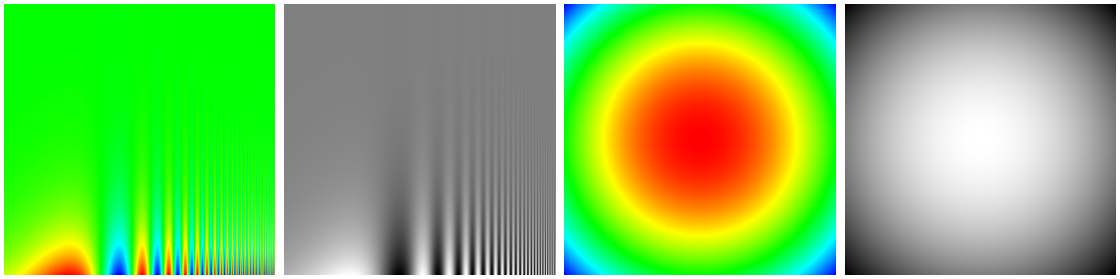


Figure 6.2. Left: the images show the color mapping of the spatial contrast sensitivity function. Frequency increases from left to right whereas contrast increases from the top to the bottom. The isoluminance of the rainbow color map obfuscate low contrast regions and small details, which can be seen using gray scale. Right: changes in color in the rainbow color map may be perceived as features in the data. The "boring" scalar field $f(x, y) = x^2 + y^2$ appears to have more features when rainbow color map is used than in the gray scale image.

color map can *obscure* data. The problem arises for data containing high spatial frequency. Isoluminant maps can obfuscate these frequencies because our visual system perceives them through changes in luminance. This is illustrated in the left images in Figure 6.2. Note how details on the top half and left portions of the rainbow color mapped image were "removed" by the choice of the color map. Lastly, the rainbow color map can also add *artifacts* to the visualization[170]. The problem is that the gradient in color map creates the illusion of patterns where none exist. This is illustrated in the right image in Figure 6.2. In association with the lack of a natural sorting order, it becomes difficult to identify that patterns are not due to the underlying data but due to the color map. Although Figure 6.2 shows simple synthetic examples, there have also been user studies and analysis showing that these problems are also present in the visualization of real world scenarios[170]. Despite its disadvantages, the rainbow color map is widely used in the sciences. In the study by Borkin *et al.*[9], participants reported that they liked it because they are "used to seeing", that the saturated colors are "easier to see", and it is the "most aesthetically pleasing". Another possible reason for its widespread use is that it is default in many popular simulation and visualization tools. Paraview is one of the tools that no longer uses the rainbow color map as the default option since the publication of Borland *et al.*'s "Rainbow Color Map (Still) Considered Harmful"[114]. The author even suggest that a better name for it would be "misleading color map".

In light of the many pitfalls of the rainbow color map, the visualization community has, in the past few years, been moving away from it. In 2005, 52% of the scientific publication using a color map at the IEEE Visualization Conference had at least one occurrence of

Table 6.2. Color maps in the AIAA journal

	Rainbow color map	Gray scale map	Other
2010	68.63%	13.73%	17.64%
2011	64.7%	15.69%	19.61%
2012	79.03%	8.65%	12.32%

the rainbow color map[10]. This number has dropped to a single paper published at the *IEEE Transactions on Visualization and Computer Graphics* in 2011. Motivated by this experiment, we reviewed all publications from the *AIAA Journal* for the years of 2010, 2011, and 2012 that contained a color map and counted the number of papers that used the rainbow color map. Table 6.2 shows the obtained results. Note that we do not evaluate the potential problems caused by the rainbow color map. Nevertheless, we tried the methodology explained above for a flow simulation dataset. The left image in Figure 6.3 shows the results of a flow simulation. Note how some regions are over emphasized (shown in red) while details are blurred (shown in green). The problems with the rainbow color map can be avoided by simply switching to another color map, such as the gray scale color map shown in the middle image in Figure 6.3. The image to the right shows the decolorized rainbow color map: although some details are easier to see, the result is still very different from the gray scale color map.

The visualization community has also investigated what should constitute a "good" color map. Research on the topic of color selection can be found in the work by Treinish *et al.* [170], Moreland[114], Kindlmann *et al.*[71], and others[97, 168]. The AIAA community can benefit from a set of standard color maps suitable for visualization of typical simulation data such as pressure fields, angle fields, *etc.*

6.2.2 Evaluation & user studies

In recent years, the Visualization community has seen a substantial increase in the number of papers dealing with evaluation of visualization techniques published within *IEEE TVCG*. Figure 6.4 shows the number of such papers published per year within the *IEEE TVCG* journal. The data was obtained by searching the *TVCG* website for the keywords "evaluation", "user study", "design study", and "case study" in articles published in the period between 2002 and 2012. We then read the abstracts to make sure the papers were indeed relevant. From this corpora, 96% of the aforementioned articles were user studies.

As a representative example, we focus on a user study by Laidlaw *et al.*[82] comparing techniques for the visualization of steady 2D vector fields. The authors recruited five experts

Figure 6.3. Velocity magnitude. Rainbow (left) and gray scale (middle) color maps were applied to a 2D flow simulation using a spectral element code for solving the incompressible Navier-Stokes Equations. Note how red regions on the rainbow color map are over emphasized while green regions "blur" details that are shown in the gray color map. The image on the right is the decolorized rainbow color map.

and 12 non-experts users to evaluate the efficacy of each of the six techniques displayed in Figure 6.5. The evaluation was measured by the user performance during the execution of several tasks of three types: critical point detection; critical points classification; and simulation of particle advection. The first two tasks are standard whereas the third task is motivated by the fact that often experts were interested in the global flow direction. The three tasks were chosen based on the authors interaction with fluid mechanics researchers. The authors built a collection of 500 vector fields for evaluation of the tasks. Among the results, they cite no significant difference between experts and non-experts regarding accuracy in the tasks or the response times. More interestingly, performance when using the standard method of arrows on a regular grid (GRID in Figure 6.5) falls below average for multiples tasks involving critical points location, classification and advection (which means that users required more time to complete the task and committed more errors). On the other end of the spectrum, user performance when using GSTR consistently scored above average. Another similar study compare the user performance when using line and tube

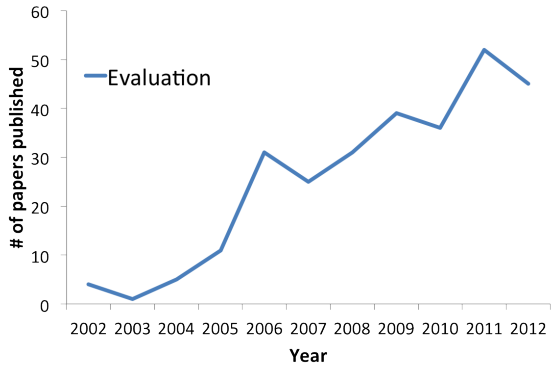


Figure 6.4. Evolution of the number of papers published on the topic of evaluation at TVCG.

integral curves (with monoscopic and stereoscopic viewing) for 3D vector field data[44]. User study can be a powerful tool for helping users choose the best tool for their needs and the visualization community has been working on evaluating and testing techniques as they become more widespread.

6.3 Uncertainty and Verification

Uncertainty visualization and visualization verification are two important topics in the pursuit for reliable visualizations. The AIAA community is familiar with both topics. In this work, however, we present some of the recent advancements in this area from the point of view of the Visualization community. The goal is to increase the user confidence in the results of the visualization by answering questions such as: how can one visualize the inherent error sources in the visualization? or, how can one increase her/his confidence that an implementation of a visualization algorithm does what was intended? In the following sections we present some of the recent developments in uncertainty visualization and the verification of isosurface extraction techniques.

6.3.1 Uncertainty visualization

In the course of scientific inquiry, uncertainty is the norm. The visualization community has recently turned its attention to uncertain data, and is trying to solve problems on how to best compute and convey uncertainty information. Since 2010, around 30 papers were published at *TVCG* on the topic, with application on information visualization and scientific visualization. So far, the community has seen several different representation for uncertainty, varying from traditional method such as bars, glyphs, and colors, to texture, multi-layering, animations, and volume rendering. At the AIAA community, we analyzed ten papers since 2010 dealing with material uncertainty, uncertainty in flows, and fluid simulation. The visualization step, on the other hand, is restricted almost exclusively to error bars and charts.

In the user study conducted by Sanyal *et al.*[144], the authors evaluate the effectiveness of four commonly used uncertainty visualization techniques: namely, glyphs size, glyphs color mapping, surface color mapping, and error bars (see Figure 6.6 for examples). The users performed two search tasks by identifying regions that are least and most uncertain, and two counting tasks where users counted the number of data and uncertainty features. The authors reported that, in general, users required more time and committed more mistakes when using error bars. The authors conjecture that a possible reasons for the poor performance displayed by error bars is due to the high density of the dataset used in

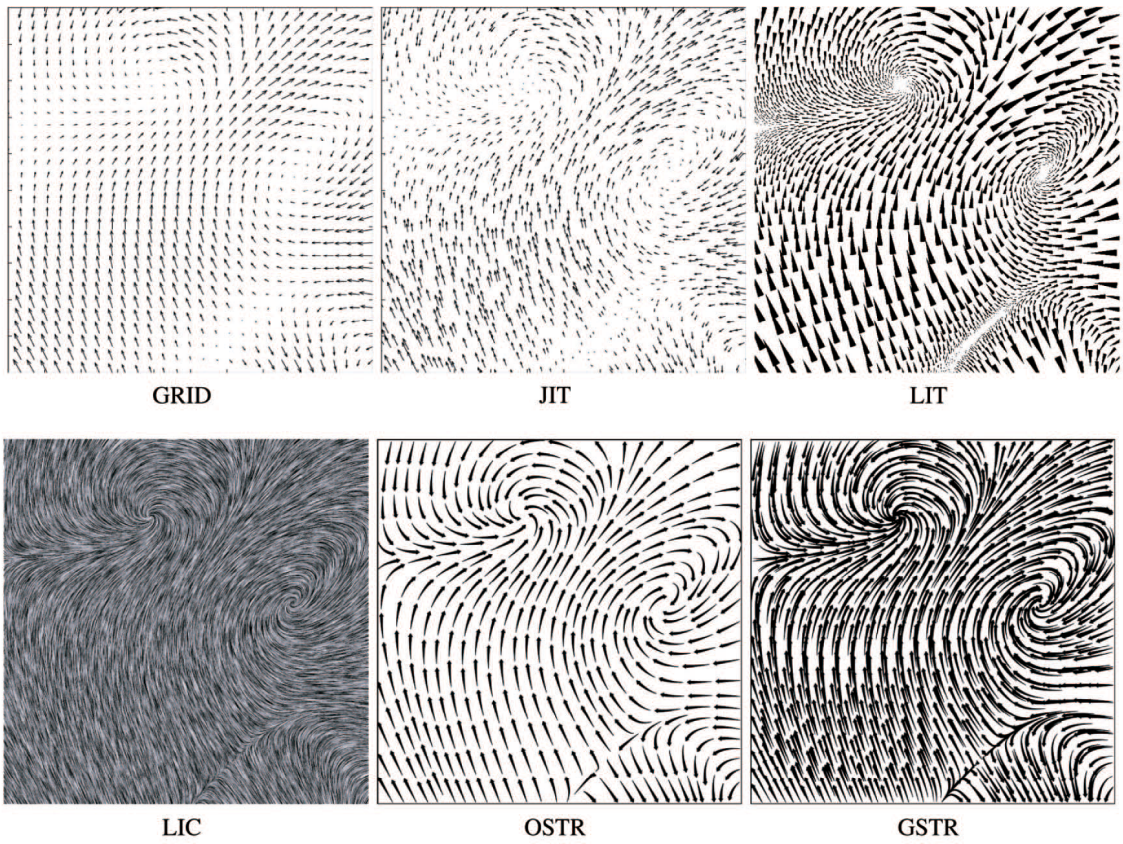


Figure 6.5. Comparing visualization methods for steady 2D vector fields. Top: standard arrow visualization, jittered arrow, icons using concepts borrowed from oil painting, respectively. Bottom: line-integral convolution, image-guided streamlines, streamlines seeded in a regular grid, respectively.

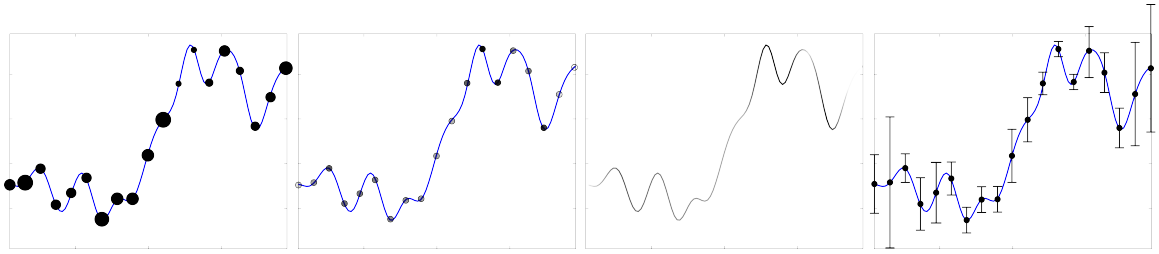


Figure 6.6. The four uncertainty visualization methods used by Sanyal *et al.*[144] in their user study. From left to right: glyphs size, glyphs color mapping, surface color mapping, and error bars.

their study. Nevertheless, a similar pattern can be found in the AIAA community (*e.g.*, see Figures 4 and 6 in Chassaing and Lucor[19]).

Several techniques for uncertainty visualization of vector fields are available. Botchen *et al.*[11] introduces a texture-mapping approach for uncertainty visualization of 2D vector

fields. Hlawatsch *et al.*[59] introduces a new static visualization of unsteady vector fields with uncertainty based on a new type of glyph. Osorio and Brodlie[1] introduce a LIC-based method for uncertainty visualization. The work by Petz *et al.*[131] uses Gaussian random fields and takes into account spatial correlation of the data, which affects vector field features. Fout and Ma[119] presents a framework based on possibility theory for uncertainty visualization and as a case study, the authors use streamlines in 3D steady vector fields. Because many researchers have recently turned their attention to uncertainty visualization, this area of research is rapidly evolving.

6.3.2 Verifiable visualization

Algorithm verification has recently attracted attention in the Visualization community. Although the need for verifying and validating image results dates back almost two decades, there is no systematic procedure for tackling the problem of verification in visualization. In particular, isosurface extraction has strong presence in AIAA journal for visualization of flow properties, and therefore, in this section we introduce two recent developments related to verification of isosurface extraction algorithm for the emerging field of verifiable visualizations.

We start our discussion on verifiable visualization by building a framework for the verification of isosurface extraction algorithms. Etiene *et al.*[42] borrowed the concept of the order of accuracy test from the CS&E community for assessing the quality of geometrical properties of isosurface extraction techniques. The authors manufacture solutions (using MMS) for which the behavior of each isosurface extraction technique could be analyzed, and then compare it against implementations. This framework requires a mathematical analysis of particular features of interest of each manufactured model in order to derive the *formal order of accuracy*, allowing one to compare the results produced computationally, *i.e.* the *observed order of accuracy*, to the one predicted by the analysis. By progressively refining the manufactured cases and analyzes and verifying that the numerical and analytical results are comparable, one can increase her/his confidence in the algorithm under scrutiny. For isosurfacing methods that generate simplicial approximations of smooth isosurfaces, the features of interest are geometric surface convergence, convergence of normals, area and curvature. By comparing numerically computed and analytical convergence rate the authors diagnoses and fixed problems within popular isosurfacing codes as well as better understand particular features of each technique, increasing the reliability on the methods under scrutiny. The practical impact of lacking of, say, area convergence is that, for some algorithms, the area error *increased* as the dataset was refined. By using a simple

manufactured solution, the authors were able to reveal bugs that prevented the convergence of some mesh properties of two publicly available isosurfacing codes.

The authors extended their work on verification of geometrical properties of isosurface extraction algorithms to the evaluation topological properties of these techniques [41]. Unlike geometry verification, topology verification cannot be performed with order-of-accuracy tests due to the discrete nature of topological properties. This renders an approach similar to *state exploration*, used in the Computer Science literature, a more appropriate route. By *exploring* different topological configurations and comparing the expected results against the obtained through the algorithm under verification, one can verify correctness of the system or find a counter-example. The authors adapted machinery from both Stratified Morse Theory[53] and Digital Topology[160] to compute surface topological invariants directly from the grid that can later be compared against those results from the isosurface extraction algorithm under verification. As an example, the authors tested an implementation of Chernyaev's marching cubes 33[21], a topologically-correct isosurface extraction algorithm, to their framework. Any implementation that preserves topology of the trilinear interpolant should be able to reproduce the case 13.5.2 of the extended marching cubes table [94]. The authors were able to find non-trivial bugs in the implementation and a non-obvious algorithm detail that is not discussed in either Chernyaev's or Lewiner's work.

6.4 Opportunities

Much of the early motivation for flow visualization in the visualization community came from the AIAA community, but over the last two decades it appears that a major gap has developed, and developments in the visualization community have been done much more independently of applications and new developments in the aeronautics area. This is in part due to the different needs of the many users of visualization techniques, including, the automotive industry, meteorology, medical imaging, geosciences, to cite a few. Summarizing decades of developments in the field of flow visualization and related areas is a nontrivial process. As an alternative, every year, a summary of recent relevant advances of visualization techniques could be published at the AIAA community; and conversely, the AIAA community could help the visualization community not only by providing expertise, but also research directions[117]. Yearly panels are held at the IEEE Vis conference, many of them with an applications focus. Consistent participation by the AIAA in these communities would help raise the level of awareness of current pressing issues. This gap between communities seems to be particular true in the need for validation and verification

of visualizations techniques and codes, which over time seem to have lost track with the new rigor expected of computational codes. A related topic is the need for increasing the level of reproducibility of computational results, which cannot be simply accomplish by making codes available to other researchers [154].

There is a natural progression from research idea within the visualization community to prototype tool, and from prototype tool to "hardened" user-available software. The challenge put forward to the visualization community to continue to seek out how to be relevant to collaborators such as our colleagues in the AIAA community, and the challenge of disseminating the advances made by the visualization community to application domains. Over the last twenty years, visualization techniques have merged as a key enabling technology for computation science by helping people explore and explain data through the creation of both static and interactive visual representations. Visualizations libraries such as Kitware's VTK contain a very large number of highly-complex visualization algorithms with thousand of lines of code implementing them. The most powerful of these algorithms are often based on complex mathematical concepts, *e.g.*, Morse-Smale complex, spectral analysis, and partial differential equations (PDEs). Robust implementations of these techniques require the use of non-trivial techniques. The overall complexity and size of these datasets leave no room for inefficient code, thus making their implementation even more complex. The complexity of the codes coupled with the new visualization techniques make it highly non-trivial for non-experts to use them, although, in principle, it should be "easier".

We believe better connections between the two communities have the chance to improve the adoption of new techniques. Furthermore, by working together, AIAA researchers can also help the Visualization community not only by providing new problems and datasets and be a major driver of problems to the community (such as they were when the visualization field was coming of age), but also by making sure the needs of the AIAA community are reflected in new research topics in Visualization.

6.5 Conclusion

In this work, we have briefly visited two decades worth of flow visualization. In particular, we first focused on vector field visualization. In this regard, we presented a classification of flow visualization seen from the perspective of the Visualization community and contrasted it with AIAA publications containing flow visualization over the last three years. By exposing the current advances in visualization, we have a starting point for

building a common research agenda that can benefit both communities. In addition, we have also visited some topics related to flow visualization that have been attracting attention in the Visualization community, namely, evaluation of visualization techniques, perception, uncertainty visualization, and verifiable visualization. The common thread in all these topics is the need for improving visualization techniques in general via error mitigation, and understanding how visualization can improve the user cognitive process. We showed some of the recent work on each of these topics in the context of flow visualization. As we mentioned at the start, (computational) flow visualization is a research area that was birthed simultaneously in two communities, and early in its development benefited from strong interaction between the communities. It is our hope that a more tight coupling between the research needs/interests of the AIAA community and the research agendas of the Visualization community can be developed. This can only happen through cooperation, collaboration and communication. In part, we hope that this work is the start of a dialog between the two communities.

Acknowledgments

The second and third authors acknowledge support by ARO W911NF-12-0375 (Program Manager Dr. Mike Coyle), NSF IIS-0914564, and Vietnam Education Foundation. The first and fourth authors acknowledge support by both NSF and DOE.

CHAPTER 7

CONCLUSION

APPENDIX A

THE COUNTEREXAMPLE IN NUMBERS

We provide the data necessary for reproducing the counterexample shown in Figure 4.5. The isosurface of interest is homeomorphic to configuration 13.5.2 of the extended Marching Cubes table.This example can be used to show that both the original and modified versions of the MC33 algorithm will fail to retrieve the correct case. Following the interior test proposed by Chernyaev, let

$$\begin{aligned} A_0 &= +0.2864 \qquad A_1 = -0.2384 \\ B_0 &= -0.0639 \qquad B_1 = +0.9486 \\ C_0 &= +0.6568 \qquad C_1 = -0.5049 \\ D_0 &= -0.1692 \qquad D_1 = +0.1075. \end{aligned}$$

The coefficient a, b, and c in $F(t)$ are given by

$$\begin{aligned} a &= +(A_1 - A_0)(C_1 - C_0) \\ &\quad -(B_1 - B_0)(D_1 - D_0) = 0.3296 \\ b &= +C_0(A_1 - A_0) \\ &\quad +A_0(C_1 - C_0) \\ &\quad -D_0(B_1 - B_0 \\ &\quad -B_0(D_1 - D_0) = -0.4886 \\ c &= A_0C_0 - B_0D_0 = 0.1701 \end{aligned}$$

Condition (i) does not hold because $a > 0$, which means that a tunnel is absent. Therefore, under Chernyaev's conditions, case 13.5.2 is incorrectly interpreted as 13.5.1.

Now, following the Lewiner's implementation, for the same scalar field, let

$$A_0 = +0.1075 \qquad A_1 = -0.5049$$
$$B_0 = -0.1692 \qquad B_1 = +0.6568$$
$$C_0 = +0.2864 \qquad C_1 = -0.0639$$
$$D_0 = -0.2384 \qquad D_1 = +0.9486.$$

The proposed alternative t is given by

$$t_{\text{alt}} = \frac{A_0}{A_0 - A_1} = 0.1756,$$

and:

$$F(t_{\text{alt}}) = -0.0007 < 0.$$

Thus condition (iii) fails, which means that case 13.5.2 is again incorrectly interpreted as 13.5.1.

APPENDIX B

AUXILIARY EXPANSIONS

In this chapter, we provide auxiliary expansions and proofs. More details on the expansions shown below can be found in the textbook by Sedgewick and Flajolet [152]. Recall that $d = D/n$.

B.1 Proof of convergence of $(1 + O(d))^n$

Let us first expand the term $(1 + C_1 x)^{C_2/x}$, where $C_1 C_2 \in \mathbb{R}^+$ and $x \to 0$.

$$\begin{aligned}
(1 + C_1 x)^{\frac{C_2}{x}} &= \exp\left(\frac{C_2}{x} \log(1 + C_1 x)\right) && \text{(B.1)} \\
&= \exp\left(\frac{C_2}{x}(C_1 x + O(x^2))\right) && \text{(B.2)} \\
&= \exp(C_1 C_2 + O(x)) && \text{(B.3)} \\
&= 1 + C_1 C_2 + O(x) = O(1) && \text{(B.4)}
\end{aligned}$$

Hence:

$$(1 + O(d))^n = (1 + O(d))^{D/d} = O(1) \tag{B.5}$$

B.2 Proof of convergence of $(1 + O(d^2))^n$

Let us first expand the term $(1 + C_1 x^2)^{C_2/x}$, where $C_1, C_2 \in \mathbb{R}^+$ and $x \to 0$.

$$\begin{aligned}
(1 + C_1 x^2)^{\frac{C_2}{x}} &= \exp\left(\frac{C_2}{x} \log(1 + C_1 x^2)\right) && \text{(B.6)} \\
&= \exp\left(\frac{C_2}{x}(C_1 x^2 + O(x^4))\right) && \text{(B.7)} \\
&= \exp(C_1 C_2 x + O(x^3)) && \text{(B.8)} \\
&= 1 + C_1 C_2 x + O(x^3) && \text{(B.9)} \\
&= 1 + O(x) && \text{(B.10)}
\end{aligned}$$

Hence:

$$(1 + O(d^2))^n = (1 + O(d^2))^{D/d} = 1 + O(d) \tag{B.11}$$

REFERENCES

[1] Allendes Osorio, R., and Brodlie, K. Uncertain flow visualization using lic. *7th EG UK Theory and Practice of Computer Graphics: Proceedings* (2009).

[2] Amtec Engineering Inc. *Tecplot, version 7 user's manual*. Amtec Engineering, Inc., 1996.

[3] Babuska, I., and Oden, J. Verification and validation in computational engineering and science: basic concepts. *Computer Aided Geometric Design Methods in Applied Mechanics and Engineering 193*, 36-38 (2004), 4057–4066.

[4] Bergner, S., Möller, T., Weiskopf, D., and Muraki, D. J. A spectral analysis of function composition and its implications for sampling in direct volume visualization. *IEEE Transactions on Visualization and Computer Graphics 12* (2006), 1353–1360.

[5] Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., and Engler, D. A few billion lines of code later: using static analysis to find bugs in the real world. *Commun. ACM 53*, 2 (2010), 66–75.

[6] Bhaniramka, P., Wenger, R., and Crawfis, R. Isosurface construction in any dimension using convex hulls. *IEEE Transactions on Visualization and Computer Aided Geometric Design Graphics 10*, 2 (2004), 130–141.

[7] Bird, D. L., and Munoz, C. U. Automatic generation of random self-checking test cases. *IBM Syst. J. 22*, 3 (1983), 229–245.

[8] Bloomenthal, J. Polygonization of implicit surfaces. *Computer Aided Geometric Design 5*, 4 (1988), 341–355.

[9] Borkin, M. A., Gajos, K. Z., Peters, A., Mitsouras, D., Melchionna, S., Rybicki, F. J., Feldman, C. L., and Pfister, H. Evaluation of artery visualizations for heart disease diagnosis. *IEEE Transactions on Visualization and Computer Graphics 17* (12/2011 2011).

[10] Borland, D., and Taylor II, R. M. Rainbow color map (still) considered harmful. *IEEE Comput. Graph. Appl. 27*, 2 (mar 2007), 14–17.

[11] Botchen, R. P., and Weiskopf, D. Texture-based visualization of uncertainty in flow fields. *Visualization Conference, IEEE 0* (2005), 82.

[12] Bowen, J. P., and Hinchey, M. G. Ten commandments of formal methods. *Computer 28*, 4 (1995), 56–63.

[13] Brambilla, A., Carnecky, R., Peikert, R., Viola, I., and Hauser, H. Illustrative flow visualization: State of the art, trends and challenges. In *EuroGraphics 2012 State of the Art Reports (STARs)* (2012), pp. 75–94.

[14] Cabral, B., and Leedom, L. C. Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), pp. 263–270.

[15] Carr, H., and Max, N. Subdivision analysis of the trilinear interpolant. *IEEE Transactions on Visualization and Computer Graphics 16* (2010), 533–547.

[16] Carr, H., Möller, T., and Snoeyink, J. Artifacts caused by simplicial subdivision. *IEEE Transaction on Visualization and Computer Aided Geometric Design Graphics 12*, 2 (2006), 231–242.

[17] Carr, H., and Snoeyink, J. Representing interpolant topology for contour tree computation. In *Topology-Based Methods in Visualization II* (2009), Springer-Verlag, pp. 59–73.

[18] Carr, H., Snoeyink, J., and Axen, U. Computing contour trees in all dimensions. *Computational Geometry: Theory and Applications 24*, 2 (2003), 75–94.

[19] Chassaing, and Lucor, D. Stochastic investigation of flows about airfoils at transonic speeds. *AIAA Journal 48* (2010), 938–950.

[20] Chen, L., and Rong, Y. Linear time recognition algorithms for topological invariants in 3d. *CoRR abs/0804.1982* (2008).

[21] Chernyaev, E. V. Marching Cubes 33: Construction of topologically correct isosurfaces. Tech. Rep. CN/95-17, High Energy Physics, 1995.

[22] Cignoni, P., Ganovelli, F., Montani, C., and Scopigno, R. Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computer Aided Geometric Designs & Graphics 24* (2000), 399–418.

[23] Cignoni, P., Ganovelli, F., Montani, C., and Scopigno, R. Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computers & Graphics 24*, 3 (2000), 399 – 418.

[24] Cignoni, P., Rocchini, C., and Scopigno, R. Metro: measuring error on simplified surfaces. *Computer Aided Geometric Design Graphics Forum 17*, 2 (1998), 167–174.

[25] Clarke, E. The birth of model checking. In *25 Years of Model Checking*, O. Grumberg and H. Veith, Eds., vol. 5000 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008, pp. 1–26.

[26] Cohen-Steiner, D., Edelsbrunner, H., and Harer, J. Stability of persistence diagrams. *Discrete and Computational Geometry 37*, 1 (2007), 103–120.

[27] Coles, P. Einstein, Eddington and the 1919 eclipse. *arXiv preprint astro-ph/0102462* (2001).

[28] De Leeuw, W., and Van Liere, R. Visualization of global flow structures using multiple levels of topology. In *Data Visualization* (1999), vol. 99, pp. 45–52.

[29] De Leeuw, W. C., and van Wijk, J. Enhanced spot noise for vector field visualization. In *Proceedings of the 6th conference on Visualization* (1995), pp. 45–52.

[30] Dey, T. K., and Levine, J. A. Delaunay meshing of isosurfaces. In *SMI '07: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007* (2007), IEEE Computer Aided Geometric Design Society, pp. 241–250.

[31] Dietrich, C., Scheidegger, C., Schreiner, J., Comba, J., Nedel, L., and Silva, C. Edge transformations for improving mesh quality of Marching Cubes. *IEEE Transaction on Visialization and Computer Aided Geometric Design Graphics 15*, 1 (2008), 150–159.

[32] Duncan, J., and Ayache, N. Medical image analysis: progress over two decades and the challenges ahead. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 1 (Jan. 2000), 85–106.

[33] Dürst, M. Letters: additional reference to marching cubes. *Computer Graphics* (1988).

[34] Edelsbrunner, H., and Harer, J. L. *Computational Topology.* American Mathematical Society, 2010.

[35] Edelsbrunner, H., and Mücke, E. P. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics 9* (1990), 66–104.

[36] Edmunds, M., Laramee, R. S., Chen, G., Max, N., Zhang, E., and Ware, C. Surface-based flow visualization. *Computers & Graphics 36*, 8 (2012), 974–990.

[37] El Hajjar, J.-F., Marchesin, S., Dischler, J.-M., and Mongenet, C. Second order pre-integrated volume rendering. In *PacificVIS '08* (2008), pp. 9–16.

[38] Engel, K., Hadwiger, M., Kniss, J. M., Lefohn, A. E., Salama, C. R., and Weiskopf, D. Real-time volume graphics. In *ACM SIGGRAPH 2004 Course Notes* (New York, NY, USA, 2004), SIGGRAPH '04, ACM, pp. 1–266.

[39] Engel, K., Hadwiger, M., Kniss, J. M., Rezk-Salama, C., and Weiskopf, D. *Real-time Volume Graphics.* A K Peters, 2006.

[40] Engel, K., Kraus, M., and Ertl, T. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Graphics Hardware Workshop* (2001), pp. 9–16.

[41] Etiene, T., Nonato, L. G., Scheidegger, C., Tierny, J., Peters, T. J., Pascucci, V., Kirby, R. M., and Silva, C. T. Topology verification for isosurface extraction. *IEEE Transactions on Visualization and Computer Graphics 18*, 6 (June 2012), 952–965.

[42] Etiene, T., Scheidegger, C., Nonato, L. G., Kirby, R. M., and Silva, C. Verifiable visualization for isosurface extraction. *IEEE Transactions on Visualization and Computer Aided Geometric Design Graphics 15*, 6 (2009), 1227–1234.

[43] Floyd, R. W. Assigning meaning to programs. In *Proceedings of the Symposium on Applied Mathematics* (1967), vol. 19, AMS, pp. 19–32.

[44] Forsberg, A., Chen, J., and Laidlaw, D. Comparing 3d vector field visualization methods: A user study. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (Nov. 2009), 1219–1226.

[45] Freire, J., T, Silva, C., Callahan, S., Santos, E., E, Scheidegger, C., and T, Vo, H. Managing rapidly-evolving scientific workflows. In *Proceedings of the International Conference on Provenance and Annotation of Data* (2006), pp. 10–18.

[46] Garth, C., Gerhardt, F., Tricoche, X., and Hans, H. Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (nov 2007), 1464–1471.

[47] Garth, C., Krishnan, H., Tricoche, X., Tricoche, T., and Joy, K. I. Generation of accurate integral surfaces in time-dependent vector fields. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (nov 2008), 1404–1411.

[48] Gelder, A. V., and Wilhelms, J. Topological considerations in isosurface generation. *ACM Transactions on Graphics 13* (1994), 337–375.

[49] Gibson, S. Using distance maps for accurate surface representation in sampled volumes. In *IEEE Volume Visualization* (1998), pp. 23–30.

[50] Globus, A., and Raible, E. Fourteen ways to say nothing with scientific visualization. *Computer 27*, 7 (July 1994), 86–88.

[51] Globus, A., and Uselton, S. Evaluation of visualization software. *SIGGRAPH 29*, 2 (1995), 41–44.

[52] Godefroid, P., Kiezun, A., and Levin, M. Y. Grammar-based whitebox fuzzing. *SIGPLAN Not. 43*, 6 (2008), 206–215.

[53] Goresky, M., and MacPherson, R. *Stratified Morse Theory*. Springer, 1988.

[54] Haller, G. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Phys. D 149*, 4 (mar 2001), 248–277.

[55] Hartmann, E. A marching method for the triangulation of surfaces. *The Visual Computer 14* (1998), 95–108.

[56] Hauser, H., Laramee, R. S., and Doleisch, H. The State of the Art in Flow visualization, part 1: Direct, Texture-based and Geometric Techniques. *IEEE Visualization* (2003).

[57] Helman, J. L., and Hesselink, L. Representation and display of vector field topology in fluid flow data sets. *Computer 22*, 8 (aug 1989), 27–36.

[58] Hildebrandt, K., Polthier, K., and Wardetzky, M. On the convergence of metric and geometric properties of polyhedral surfaces. *Geometriae Dediacata*, 123 (2006), 89–112.

[59] Hlawatsch, M., Leube, P., Nowak, W., and Weiskopf, D. Flow radar glyphs & static visualization of unsteady flow with uncertainty. *IEEE Transactions on Visualization and Computer Graphics 17*, 12 (dec. 2011), 1949–1958.

[60] Ho, C.-C., Wu, F.-C., Chen, B.-Y., Chuangs, Y.-Y., and Ouhyoungs, M. Cubical Marching Squares: Adaptive feature preserving surface extraction from volume data. *Computer Aided Geometric Design Graphics Forum 24*, 3 (2005), 537–545.

[61] Howden, W. E. Applicability of software validation techniques to scientific programs. *ACM Trans. Program. Lang. Syst. 2* (July 1980), 307–320.

[62] Hultquist, J. P. M. Constructing stream surfaces in steady 3d vector fields. In *Proceedings of the 3rd conference on Visualization '92* (Los Alamitos, CA, USA, 1992), VIS '92, IEEE Computer Society Press, pp. 171–178.

[63] IEEE. Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990* (1990), 1.

[64] Jobard, B., and Lefer, W. Creating evenly-spaced streamlines of arbitrary density. In *EG Workshop on Visualization in Scientific Computing* (1997), pp. 43–56.

[65] Jobard, B., and Lefer, W. Multiresolution flow visualization. *WSCG (Posters)* (2001), 34–35.

[66] Johnson, C. Top scientific visualization research problems. *IEEE CG&A. 24* (July 2004), 13–17.

[67] Johnson, C. R., and Sanderson, A. R. A next step: Visualizing errors and uncertainty. *IEEE CG&A. 23* (September 2003), 6–10.

[68] Ju, T. Fixing geometric errors on polygonal models: A survey. *Journal of Computer Science and Technology 24* (2009), 19–29. 10.1007/s11390-009-9206-7.

[69] Ju, T., Losasso, F., Schaefer, S., and Warren, J. Dual contouring of hermite data. *ACM Trans. Graph. 21*, 3 (July 2002), 339–346.

[70] Kindlmann, G. http://teem.sourceforge.net. Last accessed on March 24th, 2011.

[71] Kindlmann, G., Reinhard, E., and Creem, S. Face-based luminance matching for perceptual colormap generation. In *Proceedings of IEEE Visualization 2002* (October 2002), pp. 299–306.

[72] Kirby, R., and Silva, C. The need for verifiable visualization. *IEEE Computer Aided Geometric Design Graphics and Applications 28*, 5 (2008), 78–83.

[73] Kirby, R. M., Marmanis, H., and Laidlaw, D. H. Visualizing multivalued data from 2d incompressible flows using concepts from painting. In *Proceedings of the conference on Visualization '99: celebrating ten years* (Los Alamitos, CA, USA, 1999), VIS '99, IEEE Computer Society Press, pp. 333–340.

[74] Klassen, R. V., and Harrington, S. J. Shadowed hedgehogs: a technique for visualizing 2d slices of 3d vector fields. In *Proceedings of the 2nd conference on Visualization '91* (Los Alamitos, CA, USA, 1991), VIS '91, IEEE Computer Society Press, pp. 148–153.

[75] Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and Winwood, S. sel4: formal verification of an os kernel. In *SOSP '09:*

Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (New York, NY, USA, 2009), ACM, pp. 207–220.

[76] Knupp, P., and Salari, K. *Verification of Computer Codes in Computational Science and Engineering*. Chapman and Hall/CRC, 2002.

[77] Kobbelt, L., Botsch, M., Schwanecke, U., and Seidel, H.-P. Feature sensitive surface extraction from volume data. In *SIGGRAPH '01* (2001), ACM, pp. 57–66.

[78] Koop, D., Santos, E., Mates, P., Vo, H. T., Bonnet, P., Bauer, B., Surer, B., Troyer, M., N, Williams, D., E, Tohline, J., Freire, J., and T, Silva, C. A provenance-based infrastructure to support the life cycle of executable papers. *Procedia Computer Science* (2011).

[79] Kronander, J., Unger, J., Möller, T., and Ynnerman, A. Estimation and modeling of actual numerical errors in volume rendering. *Computer Graphics Forum 29*, 3 (2010), 893–902.

[80] Krüger, J., and Westermann, R. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization 2003* (2003), VIS '03, pp. 38–43.

[81] Kye, H., Shin, B., and Shin, Y. Interactive classification for pre-integrated volume rendering of high-precision volume data. *Graphical Models 70*, 6 (2008), 125–132.

[82] Laidlaw, D. H., Kirby, R. M., Jackson, C. D., Davidson, J. S., Miller, T. S., da Silva, M., Warren, W. H., and Tarr, M. J. Comparing 2d vector field visualization methods: A user study. *IEEE Transactions on Visualization and Computer Graphics 11*, 1 (Jan. 2005), 59–70.

[83] Laramee, R., Erlebacher, G., Garth, C., Schafhitzel, T., Theisel, H., Tricoche, X., Weinkauf, T., and Weiskopf, D. Applications of texture-based flow visualization. *Engineering Applications of Computational Fluid Mechanics (EACFM) 2*, 3 (2008), 264–274.

[84] Laramee, R., Hauser, H., Doleisch, H., Vrolijk, B., Post, F., and Weiskopf, D. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum 23*, 2 (2004), 203–221.

[85] Laramee, R. S. First: a flexible and interactive resampling tool for cfd simulation data. *Computers & Graphics 27*, 6 (2003), 905 – 916.

[86] Lee, J., and Newman, T. New method for opacity correction in oversampled volume ray casting. *Journal of WSCG 15* (2007), 1–8.

[87] Leeuw, D., and Van Liere, R. Divide and conquer spot noise. *Supercomputing, ACM/IEEE 1997 Conference* (1997).

[88] Lewiner, T. Personal communication. March 2010.

[89] Lewiner, T. *http://www.matmidia.mat.puc-rio.br/tomlew*, 2012 (accessed July 24, 2012).

[90] Lewiner, T., Lopes, H., Vieira, A. W., and Tavares, G. Efficient implementation of Marching Cubes' cases with topological guarantees. *Journal of Graphics Tools 8*, 2 (2003), 1–15.

[91] Li, G.-S., Tricoche, X., and Hansen, C. Gpuflic: interactive and accurate dense visualization of unsteady flows. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC conference on Visualization* (Aire-la-Ville, Switzerland, Switzerland, 2006), EUROVIS'06, Eurographics Association, pp. 29–34.

[92] Li, L., Hsieh, H., and Shen, H. Illustrative streamline placement and visualization. In *Visualization Symposium, 2008. PacificVIS'08. IEEE Pacific* (2008), IEEE, pp. 79–86.

[93] Ljung, P., Lundström, C., and Ynnerman, A. Multiresolution interblock interpolation in direct volume rendering. In *EuroVis* (2006), pp. 259–266.

[94] Lopes, A., and Brodlie, K. Improving the robustness and accuracy of the Marching Cubes algorithm for isosurfacing. *IEEE Computer Aided Geometric Design 9*, 1 (2003), 16–29.

[95] Lorensen, B. On the Death of Visualization. In *Position Papers NIH/NSF Proc. Fall 2004 Workshop Visualization Research Challenges* (2004), NIH/NSF Press.

[96] Lorensen, W., and Cline, H. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH 21* (1987), 163–169.

[97] MacDonald, L. Using color effectively in computer graphics. *Computer Graphics and Applications, IEEE 19*, 4 (1999), 20–35.

[98] Marschner, S. R., and Lobb, R. J. An evaluation of reconstruction filters for volume rendering. In *IEEE Visualization '94* (1994), pp. 100–107.

[99] MathWorks. Matlab. http://www.mathworks.com/products/matlab/. Last accessed on March 24th, 2011.

[100] Mattausch, O., Theussl, T., Hauser, H., and Gröller, E. Strategies for interactive exploration of 3d flow using evenly-spaced illuminated streamlines. In *Proceedings of the 19th spring conference on Computer graphics* (New York, NY, USA, 2003), SCCG '03, ACM, pp. 213–222.

[101] Matveyev, S. V. Marching cubes: surface complexity measure. Tech. rep., Institute for High Energy Physics, 1999.

[102] Max, N. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (1995), 99–108.

[103] McConnell, R., Mehlhorn, K., Nher, S., and Schweitzer, P. Certifying algorithms. *Computer Aided Geometric Design Science Review In Press, Corrected Proof* (2010), –.

[104] McLoughlin, T., Laramee, R. S., Peikert, R., Post, F. H., and Chen, M. Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum 29*, 6 (2010), 1807–1829.

[105] Mebarki, A., Alliez, P., and Devillers, O. Farthest point seeding for efficient placement of streamlines. In *16th IEEE Visualization Conference (VIS 2005), 23-28 October 2005, Minneapolis, MN, USA* (2005), IEEE Computer Society, p. 61.

[106] Meek, D., and Walton, D. On surface normal and gaussian curvature approximation given data sampled from a smooth surface,. *Computer Aided Geometric Design-Aided Geometric Design 17* (2000), 521–543.

[107] Meissner, M., Huang, J., Bartz, D., Mueller, K., and Crawfis, R. A practical evaluation of popular volume rendering algorithms. In *IEEE Volume Visualization* (2000), pp. 81–90.

[108] Meroney, R. Wind tunnel and numerical simulation of pollution dispersion: a hybrid approach. *Wind Engineering and Fluid Laboratory, Colorado State University, Fort Collins, USA. Invited Lecture, Croucher Advanced Study Institute, Hong Kong University of Science and Technology, 6th-10th December* (2004).

[109] Meyer-Spradow, J., Ropinski, T., Mensmann, J., and Hinrichs, K. H. Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations. *IEEE CG&A. 29*, 6 (Nov./Dec. 2009), 6–13.

[110] Möller, T., Machiraju, R., Mueller, K., and Yagel, R. Classification and local error estimation of interpolation and derivative filters for volume rendering. In *IEEE Volume Visualization* (1996), pp. 71–78.

[111] Möller, T., Machiraju, R., Mueller, K., and Yagel, R. Evaluation and design of filters using a Taylor series expansion. *IEEE Transactions on Visualization and Computer Graphics 3* (April 1997), 184–199.

[112] Montani, C., Scateni, R., and Scopigno, R. A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer: International Journal of Computer Graphics* (1994).

[113] Montani, C., Scateni, R., and Scopigno, R. A modified look-up table for implicit disambiguation of Marching Cubes. *The Visual Computer Aided Geometric Design 10*, 6 (December 1994), 353–355.

[114] Moreland, K. Default color map. *http://www.org//ParaView3//index.php//Default_Color_Map* (2007).

[115] Moreland, K., and Angel, E. A fast high accuracy volume renderer for unstructured data. In *IEEE Volume Visualization and Graphics* (2004), pp. 9–16.

[116] Munkres, J. R. *Topology, A First Course*. Prentice-Hall, Inc., 1975.

[117] Munzner, T., Johnson, C., Moorhead, R., Pfister, H., Rheingans, P., and Yoo, T. S. Nih-nsf visualization research challenges report summary. *IEEE Comput. Graph. Appl. 26*, 2 (Mar. 2006), 20–24.

[118] Natarajan, B. K. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer 11*, 1 (Jan. 1994), 52–62.

[119] Nathaniel Fout, K.-l. M. Reliable visualization: Verification of visualization based on uncertainty analysis. Tech. rep., University of California, Davis, 2012.

[120] Newman, T. S., and Yi, H. A survey of the Marching Cubes algorithm. *Computer Aided Geometric Designs & Graphics 30*, 5 (2006), 854–879.

[121] Nielson, G. M. On Marching Cubes. *IEEE Computer Aided Geometric Design 9*, 3 (2003), 283–297.

[122] Nielson, G. M., and Hamann, B. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceedings of the 2nd conference on Visualization* (Los Alamitos, CA, USA, 1991), IEEE Computer Society Press, pp. 83–91.

[123] Ning, P., and Bloomenthal, J. An evaluation of implicit surface tilers. *IEEE Computer Aided Geometric Design Graphics and Applications 13*, 6 (1993), 33–41.

[124] Novins, K., and Arvo, J. Controlled precision volume integration. In *ACM Volume Visualization* (1992), pp. 83–89.

[125] Palacios, J., and Zhang, E. Interactive visualization of rotational symmetry fields on surfaces. *IEEE Transactions on Visualization and Computer Graphics 17* (2011), 947–955.

[126] Pascucci, V., and Cole-McLaughlin, K. Parallel computation of the topology of level sets. *Algorithmica 38*, 1 (2003), 249–268.

[127] Patera, J., and Skala, V. A comparison of fundamental methods for iso surface extraction. *Machine Graphics & Vision International Journal 13*, 4 (2004), 329–343.

[128] Peikert, R., and Sadlo, F. Topologically relevant stream surfaces for flow visualization. In *Proceedings of the 2009 Spring Conference on Computer Graphics* (New York, NY, USA, 2009), SCCG '09, ACM, pp. 35–42.

[129] Peng, Z., Grundy, E., Laramee, R. S., Chen, G., and Croft, N. Mesh-driven vector field clustering and visualization: An image-based approach. *IEEE Transactions on Visualization and Computer Graphics 18*, 2 (feb 2012), 283–298.

[130] Peng, Z., and Laramee, R. Higher dimensional vector field visualization: A survey. In *Theory and Practice of Computer Graphics* (2009), The Eurographics Association, pp. 149–163.

[131] Petz, C., Pthkow, K., and Hege, H.-C. Probabilistic local features in uncertain vector fields with spatial correlation. *Computer Graphics Forum 31*, 3pt2 (2012), 1045–1054.

[132] Pobitzer, A., Peikert, R., Fuchs, R., Schindler, B., Kuhn, A., Theisel, H., Matkovi?, K., and Hauser, H. The state of the art in topology-based visualization of unsteady flow. *Computer Graphics Forum 30*, 6 (2011), 1789–1811.

[133] Pommert, A., and Höhne, K. H. Evaluation of image quality in medical volume visualization: The state of the art. In *Proceedings of the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II* (London, UK, 2002), MICCAI '02, Springer-Verlag, pp. 598–605.

[134] Pommert, A., and Höhne, K. H. Validation of medical volume visualization: a literature review. *International Congress Series 1256* (2003), 571–576. CARS 2003. Computer Assisted Radiology and Surgery. Proceedings of the 17th International Congress and Exhibition.

[135] Pommert, A., Tiede, U., and Höhne, K. On the accuracy of isosurfaces in tomographic volume visualization. In *MICCAI'02* (London, UK, 2002), Springer-Verlag, pp. 623–630.

[136] Popper, K. R. *The Logic of Scientific Discovery*. Routledge, 2002. 1st English Edition:1959.

[137] Raman, S., and Wenger, R. Quality isosurface mesh generation using an extended Marching Cubes lookup table. *Computer Graphics Forum 27*, 3 (2008), 791–798.

[138] Riley, T., Goucher, A., Tim, R., and Adam, G. *Beautiful Testing: Leading Professionals Reveal How They Improve Software*, 1st ed. O'Reilly Media, Inc., 2009.

[139] Roache, P. J. *Verification and Validation in Computational Science and Engineeering*. Hermosa Publisher, 1998.

[140] Röttger, S., Guthe, S., Weiskopf, D., Ertl, T., and Strasser, W. Smart hardware-accelerated volume rendering. In *VISSYM '03* (2003), pp. 231–238.

[141] Röttger, S., Kraus, M., and Ertl, T. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *IEEE Visualization* (2000), pp. 109–116.

[142] Roy, C. J. Review of code and solution verification procedures for computational simulation. *J. Comput. Phys. 205*, 1 (2005), 131–156.

[143] Sakkalis, T., Peters, T. J., and Bisceglio, J. Isotopic approximations and interval solids. *Computer Aided Geometric Design-Aided Design 36*, 11 (2004), 1089–1100.

[144] Sanyal, J., Zhang, S., Bhattacharya, G., Amburn, P., and Moorhead, R. A user study to compare four uncertainty visualization methods for 1d and 2d datasets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (Nov. 2009), 1209–1218.

[145] Schaefer, S., Ju, T., and Warren, J. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (May 2007), 610–619.

[146] Schafhitzel, T., Tejada, E., Weiskopf, D., and Ertl, T. Point-based stream surfaces and path surfaces. In *Proceedings of Graphics Interface 2007* (New York, NY, USA, 2007), GI '07, ACM, pp. 289–296.

[147] Scheidegger, C., Etiene, T., Nonato, L. G., and Silva, C. Edge flows: Stratified Morse Theory for simple, correct isosurface extraction. Tech. rep., University of Utah, 2010.

[148] Scheuermann, G., Krüger, H., Menzel, M., and Rockwood, A. P. Visualizing nonlinear vector field topology. *IEEE Transactions on Visualization and Computer Graphics 4*, 2 (apr 1998), 109–116.

[149] Schreiner, J., Scheidegger, C., and Silva, C. High-quality extraction of isosurfaces from regular and irregular grids. *IEEE Transactions on Visualization and Computer Aided Geometric Design Graphics 12*, 5 (2006), 1205–1212.

[150] Schreiner, J., Scheidegger, C. E., Fleishman, S., and Silva, C. T. Direct (re)meshing for efficient surface processing. *Computer Graphics Forum 25*, 3 (2006), 527–536.

[151] Schroeder, W., Martin, K., and Lorensen, W. *Visualization Toolkit, An Object-Oriented Approach to 3D Graphics - 2nd ed.* Prentice-Hall, 1998.

[152] Sedgewick, R., and Flajolet, P. *Introduction to Analysis of Algortithms*, 1st ed. Addison-Wesley Professional, 1995.

[153] Seger, C. An introduction to formal hardware verification. Tech. rep., University of British Columbia, Vancouver, BC, Canada, Canada, 1992.

[154] Silva, C. T., Freire, J., and Callahan, S. P. Provenance for visualizations: Reproducibility and beyond. *Computing in Science and Engineering. 9*, 5 (Sept. 2007), 82–89.

[155] Silva, C. T., and Mitchell, J. S. B. Greedy cuts: an advancing front terrain triangulation algorithm. In *Proceedings of the 6th ACM international symposium on Advances in geographic information systems* (New York, NY, USA, 1998), GIS '98, ACM, pp. 137–144.

[156] Silva, S., Santos, B. S., and Madeira, J. Using color in visualization: A survey. *Computers & Graphics 35*, 2 (2011), 320 – 333.

[157] Smelyanskiy, M., Holmes, D., Chhugani, J., Larson, A., Carmean, D. M., Hanson, D., Dubey, P., Augustine, K., Kim, D., Kyker, A., Lee, V. W., Nguyen, A. D., Seiler, L., and Robb, R. Mapping high-fidelity volume rendering for medical imaging to CPU, GPU and many-core architectures. *IEEE Transactions on Visualization and Computer Graphics 15* (November 2009), 1563–1570.

[158] Spencer, B., Laramee, R., Chen, G., and Zhang, E. Evenly spaced streamlines for surfaces: An image-based approach. *Computer Graphics Forum 28*, 6 (2009), 1618–1631.

[159] Squillacote, A. *The ParaView Guide: A Parallel Visualization Application.* Kitware, 2007.

[160] Stelldinger, P., Latecki, L. J., and Siqueira, M. Topological equivalence between a 3D object and the reconstruction of its digital image. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 1 (2007), 126–140.

[161] Sutton, P., Hansen, C., Shen, H.-W., and Schikore, D. A case study of isosurface extraction algorithm performance. In *Data Visualization 2000* (2000), Springer, pp. 259–268.

[162] Taubin, G., Cukierman, F., Sullivan, S., Ponce, J., and Kriegman, D. Parameterized families of polynomials for bounded algebraic curve and surface fitting. *IEEE PAMI 16*, 3 (Mar 1994), 287–303.

[163] Theisel, H. Exact isosurfaces for marching cubes. *Computer Graphics Forum 21*, 1 (2002), 19–32.

[164] Theisel, H., Rssl, C., and Seidel, H.-P. Compression of 2d vector fields under guaranteed topology preservation. *Computer Graphics Forum 22*, 3 (2003), 333–342.

[165] Theisel, H., Rssl, C., and Weinkauf, T. Topological representations of vector fields. In *Shape Analysis and Structuring*, L. Floriani and M. Spagnuolo, Eds., Mathematics and Visualization. Springer Berlin Heidelberg, 2008, pp. 215–240.

[166] Theisel, H., Weinkauf, T., Hege, H.-C., and Seidel, H.-P. Topological methods for 2d time-dependent vector fields based on stream lines and path lines. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (jul 2005), 383–394.

[167] Tohline, J. E., and Santos, E. Visualizing a Journal that Serves the Computational Sciences Community. *Computing in Science & Engineering 12*, 3 (2010), 78–81.

[168] Tominski, C., Fuchs, G., and Schumann. Task-Driven Color Coding. In *Information Visualisation, 2008. IV '08. 12th International Conference* (2008), pp. 373–380.

[169] Tory, M., and Möller, T. Human factors in visualization research. *IEEE Transactions on Visualization and Computer Graphics 10*, 1 (Jan. 2004), 72–84.

[170] Treinish, L., et al. Why should engineers and scientists be worried about color? *IBM Thomas J. Watson Research Center, Yorktown Heights, NY* (2009).

[171] Turk, G., and Banks, D. Image-guided streamline placement. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 453–460.

[172] Ueng, S.-K., Sikorski, C., and Ma, K.-L. Efficient streamline, streamribbon, and streamtube constructions on unstructured grids. *IEEE Transactions on Visualization and Computer Graphics 2*, 2 (jun 1996), 100–110.

[173] Unser, M. Splines: a perfect fit for signal and image processing. *Signal Processing Magazine, IEEE 16*, 6 (nov 1999), 22 –38.

[174] Usman, A., Möller, T., and Condat, L. Gradient estimation revitalized. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1495–1504.

[175] van Gelder, A., and Wilhelms, J. Topological considerations in isosurface generation. *ACM Transactions on Graphics 13*, 4 (1994), 337–375.

[176] Van Wijk, J. Spot noise texture synthesis for data visualization. *ACM SIGGRAPH Computer Graphics* (1991).

[177] Varadhan, G., Krishnan, S., Kim, Y. J., and Manocha, D. Feature-sensitive subdivision and isosurface reconstruction. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), VIS '03, IEEE Computer Society, pp. 14–.

[178] Weber, G. H., Scheuermann, G., Hagen, H., and Hamann, B. Exploring scalar fields using critical isovalues. In *Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), IEEE Computer Aided Geometric Design Society, pp. 171–178.

[179] Weinkauf, T., and Theisel, H. Streak lines as tangent curves of a derived vector field. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (nov 2010), 1225–1234.

[180] Weinkauf, T., Theisel, H., Shi, K., Hege, H.-C., and Seidel, H.-P. Extracting higher order critical points and topological simplification of 3D vector fields. In *Proc. IEEE Visualization 2005* (Minneapolis, U.S.A., October 2005), pp. 559–566.

[181] Westover, L. Interactive volume rendering. In *ACM Volume Visualization* (1989), pp. 9–16.

[182] Wiebel, A., and Scheuermann, G. Eyelet particle tracing-steady visualization of unsteady flow. In *Visualization, 2005. VIS 05. IEEE* (2005), IEEE, pp. 607–614.

[183] Williams, P. L., and Max, N. A volume density optical model. In *ACM Volume Visualization* (1992), pp. 61–68.

[184] Williams, P. L., Max, N. L., and Stein, C. M. A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics 4* (January 1998), 37–54.

[185] Xu, G. Convergence analysis of a discretization scheme for gaussian curvature over triangular surfaces. *Computer Aided Geometric Design 23*, 2 (2006), 193–207.

[186] Xu, Z., Xu, G., and Sun, J.-G. Convergence analysis of discrete differential geometry operators over surfaces. In *Mathematics of Surfaces XI*, vol. 3604 of *LNCS*. Springer, 2005, pp. 448–457.

[187] Yang, J., Twohey, P., Engler, D., and Musuvathi, M. Using model checking to find serious file system errors. *ACM Trans. Comput. Syst. 24* (November 2006), 393–423.

[188] Zhang, N., Hong, W., and Kaufman, A. Dual contouring with topology-preserving simplification using enhanced cell representation. In *Proceedings of the conference on Visualization '04* (Washington, DC, USA, 2004), VIS '04, IEEE Computer Society, pp. 505–512.

[189] Zheng, Z., Xu, W., and Mueller, K. VDVR: Verifiable volume visualization of projection-based data. *IEEE Transactions on Visualization and Computer Graphics 65*, 6 (2010), 1515–1524.

[190] Zhou, L., and Pang, A. Metrics and visualization tools for surface mesh comparison. In *Proc. SPIE - Visual Data Exploration and Analysis VIII* (2001), vol. 4302, pp. 99–110.