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:

First M. Last , Chair enter date

Date Approved

First M. Last , Member

Date Approved

First M. Last , Member

Date Approved

ABSTRACT

Abstract text here.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Acknowledgement text here.

CHAPTER 1

INTRODUCTION

The ever increasing capability of acquiring data has provided more tools to scientists for observing phenomena, hypothesis creation, and ultimately, derive, expand, or correct scientific theories. Simultaneously, the field of scientific visualization emerged and gained importance for it became the means through which scientists explore and evaluate their results. Techniques falling under the scope of scientific visualization – isosurface extraction, direct volume rendering, topological analysis, flow visualization, among others – have successfully been used in a variety of fields, including medical diagnosis, computational fluid dynamics, weather simulation, among others. The uniqueness of each field and the nature of the data of each has motivated the development of complex visualization tool. With the increasing complexity and importance of the visualization techniques in the scientific pipeline, the *reliability* of visualization is of great importance for the data analysis. By "reliable", we include many topics of current interest of the visualization community that contributes to increase the user's confidence in the images generated. These topics include uncertainty visualization, uncertainty propagation, evaluation, user perception, and verification.

The main contribution of this thesis is to advance the theory and practice of *verification of visualization algorithms and implementation*. The definition of what we mean by verification of visualization techniques will become clear in the next sections and chapters. We provide the theoretical background for the verification of visualization algorithms and the results of applying these techniques to several available visualization implementations. Before we dive into the details and obtained results, we will go over the motivations behind verifying visualizations. At the heart of the methodology for verifying visualization lies the well-known scientific method.

1.1 The scientific method

New theories are put forward and evaluated through the scientific method: observations; hypothesis creation; 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*. From "The Logic of Scientific Discovery" [93]:

> 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.

In other words, falsification is the process by which the theory risky predictions are tested, which is not the same as to repeatedly prove its correctness.

An example of the testing risky prediction is the classic Eddington's experiment of Einstein's theory of relativity [21]. Eddington conducted an experiment in order to measure of light bending caused by the massive size of the Sun. During the eclipse of 29 May 1919, Eddington photographed the Hyades star cluster so as to measure the amount of bending of the light coming from that group of stars. 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) deviations did not matched the observed deviation, in which case the theory would be refuted; or, the deviation 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.

1.2 The weakest link

The sensitivity of a theory thus requires that given statement to be carefully tested: problems during the test phase may cause the theory to be wrongly convicted of falsity. Thus, the scientific community have developed several ways of increasing the reliability of their tests: statistic: powerful statistical methods have been developed; mathematical models were improved; computational models have been built, among others. Each of the tools cited previously, is, in itself, has is also critically verified. Theorems have to be revised; statistical methods needs to be improved; computational models must be verified. Thus, the reliability of scientific theories is directly related to the methodology used. In many of the sub steps involved in the creation of a scientific theory, the scientific methodology can be applied in order to increase our confidence that an individual part of the method is correct. Clearly, this is not always the case: a theorem can be proven to be true or false. In other cases, one needs to resort to the testability of some statement. The case of *code-an-solution-verification* is one of them. In computational science, code verification is concerned with the reliability of the solutions provided by a numerical code. Since there code can be proven correct in only some cases [**?**], often the goal is to increase the reliability that a certain statement about the code is true. If the tests are sufficiently thorough, the confidence that a particular code is corrects increases. In the same way that the computation science community has developed new ways of increase confidence in their code, the computer science community has also developed techniques appropriated for their need for increasing the confidence that certain statements about the software correctness are true.

Now, let us revisit the simplified traditional scientific pipeline and see how rigorously all the steps have been taken. As one can see, all but the last step, the visualization step, have rigorously, well-defined and widely accepted tools for increase the chances that that step has not introduced undue error. In this thesis, we present techniques for increasing one's confidence that certain claims about visualization algorithm are indeed true.

I need to write a paragraph or two about the more general reliability of visualization algorithms. This purpose of this thesis is to advance the state of reliability of visualization and thus stye overall traditional scientific pipeline. I should also mention that different techniques have to be applied. I shall mention that every single visualization technique should be verified.

In this age of scientific computing, the simulation science pipeline of mathematical modeling, simulation and evaluation is a rendition of the scientific method as commonly

employed as the traditional experimental pipeline. Critical to this simulation approach is the evaluation stage in which numerical data are post-processed, visualized and then examined in order to answer the original queries that instigated the investigation. In fact, visualization of scientific results has become as much a part of the scientific process as mathematical modeling or numerical simulation.

Despite its growing importance in the scientific computing process, 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.

In this work, we advocate the use of techniques from the *verification and validation* process used in the engineering community. While the lack of a well-established framework for verifying visualization tools has meant that a variety of analysis techniques have been developed [127, 114], we believe that visualization has achieved sufficient importance to warrant investigation of stricter verification methodologies. Several authors have already asserted this need [36, 50], and in this work we present techniques that are a concrete step towards reducing the gap between best practices in simulation science and visualization.

The purpose of this work is to present a verification methodology for visualization tools that has comparable rigor to that of other components of the computational scientific pipeline. More specifically, we set out to define a verification methodology in the mold of the area of V&V, in the context of visualization. Furthermore, we illustrate our proposal by testing several publicly available isosurface extraction codes to the verification procedure, giving a detailed description of the steps involved in the process.

It is important to emphasize that the point of verification procedures is not to compare algorithms to one other with the hope of finding the best alternative. This procedure equips developers of new algorithms and/or implementations with a process that provides a systematic way of identifying and correcting errors in both algorithms and implementations. The goal is to provide users with a methodology that will give them a more concrete model for the behavior of the implementation, which will increase confidence in the visualization tools. As we will show, a fair verification analysis can bring out unforeseen behavior, and quickly detect implementation problems that might not be caught through standard

comparisons or visual inspection.

The contributions of this work are threefold. To the best of our knowledge, we, for the first time, apply the framework of verification to assess the correctness of visualization tools. Furthermore, we provide a detailed description of how to accomplish the verification procedure by subjecting different isosurfacing tools to the battery of tests comprising the V&V process. Our second contribution is the underlying mathematical analysis and associated manufactured solutions developed to analyze the isosurfacing methods. We should clarify that when applying MMS for other techniques (even in the case of isosurface extraction), the theoretical analysis should be tailored to the particular features of these algorithms.

The manufactured solutions presented here are simple but general enough to be promptly employed for evaluating other visualization tools besides isosurfacing. Our third contribution is a comprehensive set of results obtained using the technique, including the finding of implementation errors in two publicly available isosurface extraction codes.

1.3 Verification in CS and CS&E

We describe the components of the verification pipeline shown in Figure **??** (left). 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" [**?**]. 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[**?**], model checking[**?**], fuzzing[**?**, **?**], 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 that must be verified, verification can be considered as a process where one accumulates evidence that a code is correct[**?**] instead of deriving a proof that the code is actually correct. These techniques have been successfully applied not only for verification of user-level computer code[**?**], but also hardware[**?**] and operating system kernel[**?**].

The 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 (middle diagram in Figure **??**). 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 of computational, should be refined. A successful approach for code verification, known to be sensitive to code mistakes[**?**], is the order of accuracy test — an evaluation of the implementation behavior when submitted to successive grid refinement [**?**]. In this scenario, one can use an analytical or a manufactured solutions for the PDE. An analytical solution is an actual solution for the PDE while manufactured solutions are exact solutions of a slightly modified version (in terms of boundary conditions or forcing functions) of the original PDE. In the order of accuracy test, the analytical or manufactured solution is used to extract the observed behavior (or observed order of accuracy) which is then compared to the expected behavior (or formal order of accuracy).

Babuška and Oden define verification as "the process of determining if a computational model obtained by discretizing a mathematical model 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]. Although they review the concept only in the context of computational science and engineering applications, it is important to appreciate that the same idea applies to scientific visualization. *Verification is about investigating to what extent a (numerical) approximation scheme – both in algorithm and corresponding implementation – represents the desired mathematical model.* Validation, on the other hand, is about ensuring that the model represents physical reality. In this paper, we will concern ourselves only with verification, under the assumption that the model has been validated by the user of the technique. This is the perspective Kirby and Silva suggest for "Verifiable Visualization" [50].

One of the main requirements for verifiable visualization is to have a rigorous analysis which predicts the results of the algorithm and its implementation when evaluating it on a known model problem. The more complete this analysis is, the more thorough the testing procedures can be. This *continuous process* of verification through refinement of key controllable input parameters of the method (such as grid spacing) and testing is different from a one-shot process.

The verification process should involve a suite of tests with corresponding results from which one can progressively increase reliance on the method under analysis. When appro-

priately applied, verification provides ways to appreciate the nuances of the applicability of the method. As we will see in this paper, writing down the analysis for the expected result of isosurface extraction gives us concrete bounds on what features we can expect the resulting surface to have, and these are extremely important for users.

A common practice in the visualization community is to test implementations by using complicated, "real-world" datasets. The value of these tests is that they provide evidence of the algorithm's applicability. We advocate a complementary approach: developers should carefully manufacture test cases that can be mathematically modeled and analyzed, called *manufactured solutions*. These manufactured solutions can then be used to test the implementations. In this paper, we present analysis that describe the expected rate of convergence of several isosurface features, and test implementations acting on our model problems using simple analytical volumes. As we show in Sections 2.2 and 2.3, this method helps pin down the mathematical characteristics of the technique, and, more practically, it is quite effective at uncovering implementation bugs.

The challenge behind manufactured solutions is to construct them in a way that allows us to predict the expected behavior of the method under investigation. Moreover, the manufactured solutions should tax the method vigorously, bringing out potential problems. In Section 2.4, we will present some situations where incorrectly chosen manufactured solutions have a big impact in the results. We do so to emphasize that all components of the pipeline, even the construction and evaluation of the manufactured solutions, must be meticulously handled to maintain the rigor of the verification process.

1.4 Verification in Visualization

The costs associated with not verifying software are mind-boggling (see report in Desktop/ folder); Talk about colorful fluids dynamics.

1.5 Thesis organization

This thesis is organized as follows: Chapter 2, 3, and 5 present the theory used for verifying visualization algorithms and provide the results for of applying the verification to well-known and widely used visualization algorithms and implementations. Chapter 4 shows how the verification procedure developed in Chapter 3 helped us uncover an 18 years old problem in a topologically correct isosurface extraction algorithm. We explain the details of the problem and provide the solution for it. In chapter **??**, we detail some of the recent advances in techniques for flow visualization and show some studies on the topic of reliability of visualization algorithms. The goal is to present to a client community, in our

case the AIAA community, some of the cutting edge results that can help them increase reliability of their implementation.

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 [83, 60], mesh quality [25, 101], accuracy [86, 127], and performance [108]. The influence of different reconstruction schemes and filters in scalar visualization has also been examined [12, 92]. 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 [67]. In the context of isosurface extraction, researchers routinely use tools such as Metro [19] 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 [36] and more recently Kirby and Silva [50] 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

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

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 Figures 1 and 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 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 [95, 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 [73, 125, 126, 41]. 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 $|f(\vec{v}) - \lambda|$ [109]. Notice that algebraic distances only makes

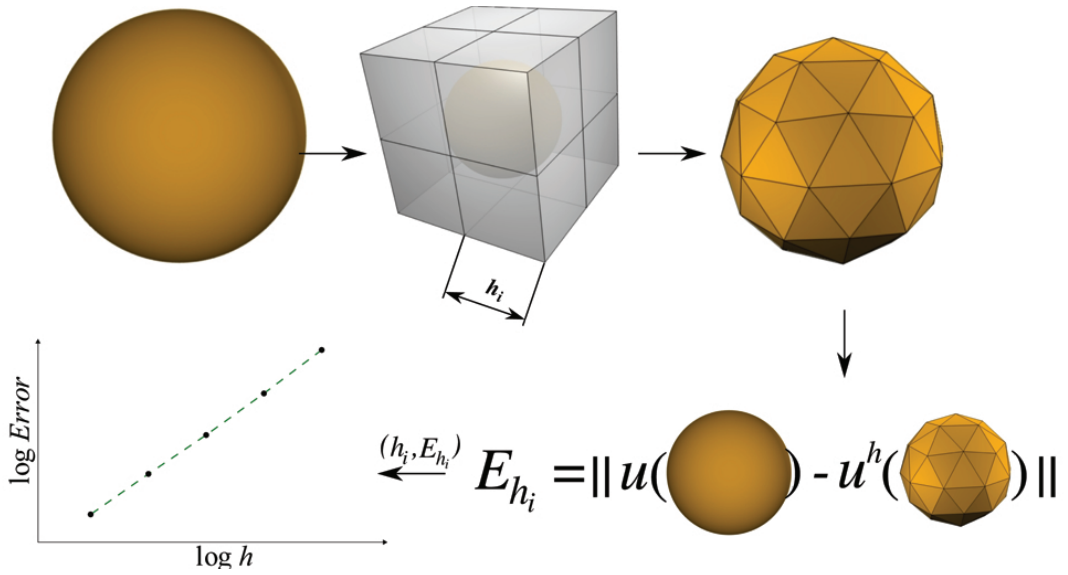


Figure 2.1. Workflow for the method of manufactured solution (Figure 1), clockwise from the top left.

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, the geometric distance between $\vec{v}$ and S is approximated by $|f(\vec{v}) - \lambda|/|\nabla f(\vec{v})|$ [109]. 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

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

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

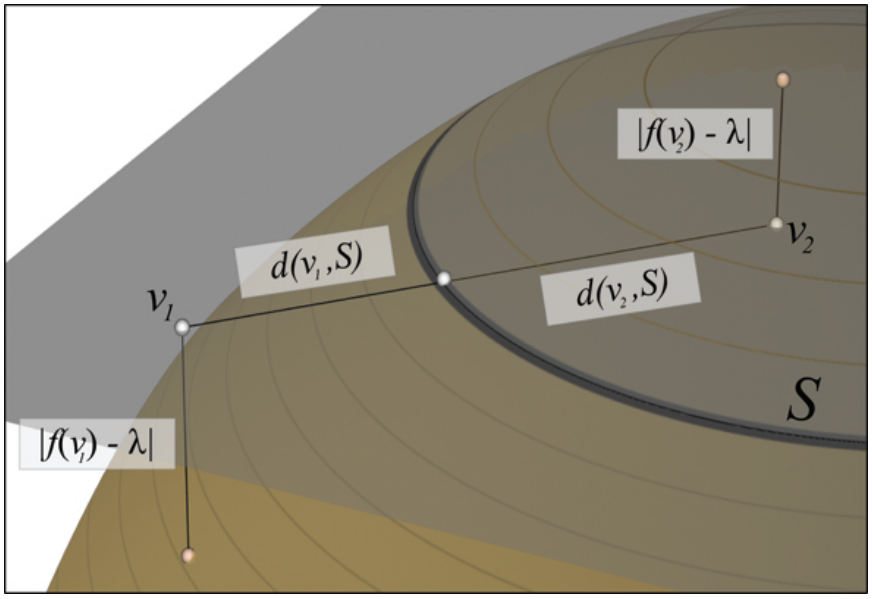


Figure 2.2. The distance between a point $\vec{v}$ and the isosurface S with isovalue λ can be approximated by the algebraic distance divided by the gradient magnitude of the scalar field at $\vec{v}$, $|f(\vec{v}) - \lambda|/|\nabla f(\vec{v})|$. In the figure, the thick circle represents the isosurface 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.

$g(x(0,0), y(0,0)) = 0$, 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 $(x(0,0), y(0,0)) = (x_i, y_j)$).

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 $p_1 p_2 p_3$ can be computed by:

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

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 $\vec{n_{p_1 p_2 p_3}}$ becomes:

$$\vec{n_{p_1 p_2 p_3}} = r h^2 \left(-\frac{\partial g}{\partial x} + O(h), -\frac{\partial g}{\partial y} + O(h), 1 \right)^T \tag{2.8}$$

where $r = u_2 v_3 - u_3 v_2$. Comparing the exact normal vector $\vec{n_0}$ in Equation 2.6 with $\vec{n_{p_1 p_2 p_3}}$ 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.* [41]. However, the authors only establish asymptotic convergence, with no order of accuracy associated with it. The argument is more mathematically involved than space

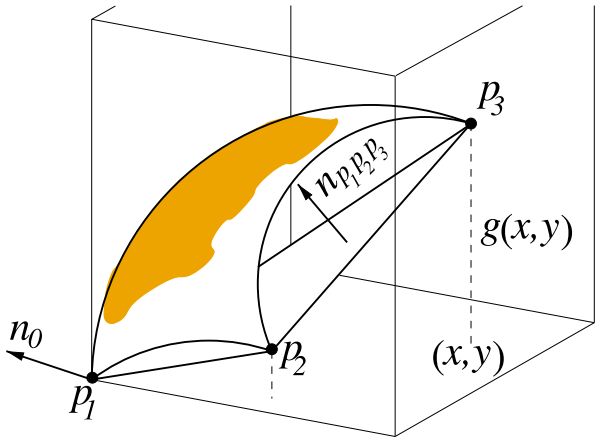


Figure 2.3. Isosurface local parametrization and approximation.

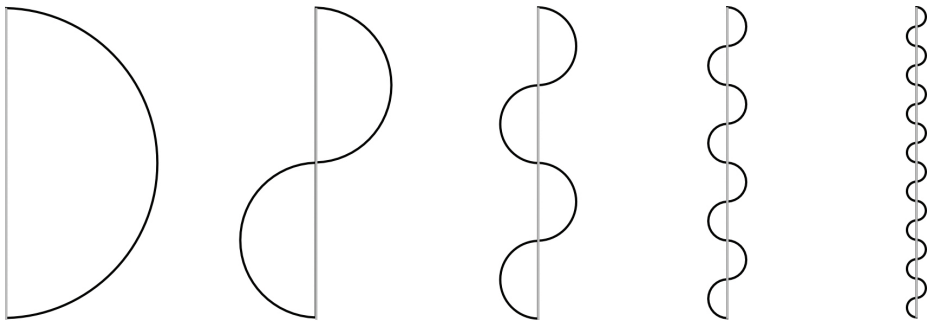


Figure 2.4. Uniform convergence does not imply convergence in area. The sequence of curves converges uniformly to a straight line, but the length of the curves does not change.

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) [73]. Meek and Walton [73] 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 [125] 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 [73] 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 [65], SnapMC [94], Macet [25], Dual Contouring [48], Afront [101], and DelIso [24].

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 [65] (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 [94] 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.* [4]. 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 [25] 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. Therefore, some corners of the background grid are displaced so as to avoid the parallel-like intersections.

Dual Contouring: Dual Contouring [48] 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 [53] and SurfaceNets [35] 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

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

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 [101] 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 [24] 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.

	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).

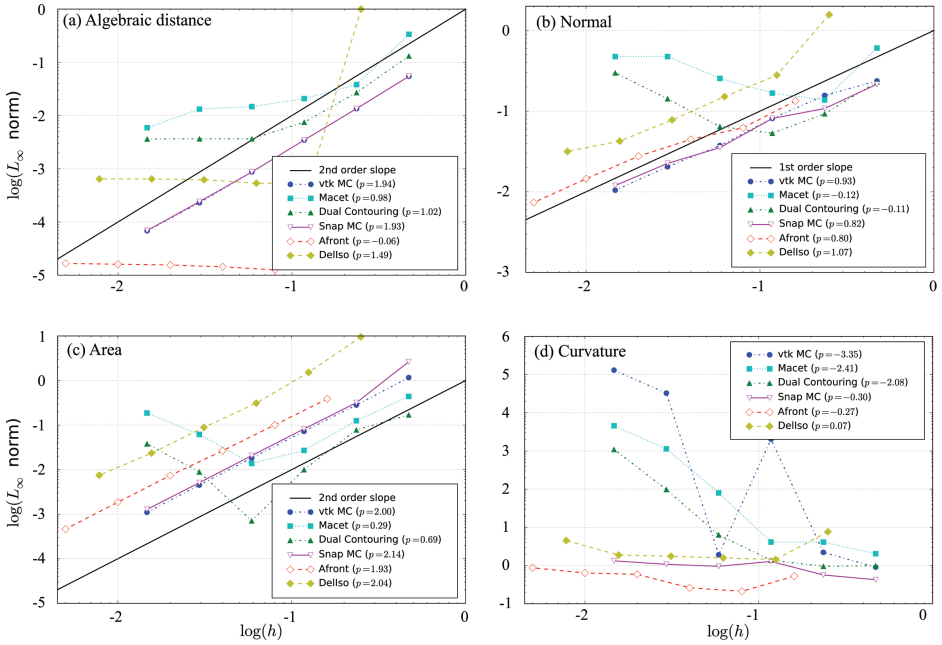


Figure 2.5. Observed order of accuracy. The implementations of Macet and Dual Contouring have a bug that causes the deviation on errors. The black continuous line represents the expected behavior. p is the slope of the linear regression for each curve.

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 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:

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

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 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 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,

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.

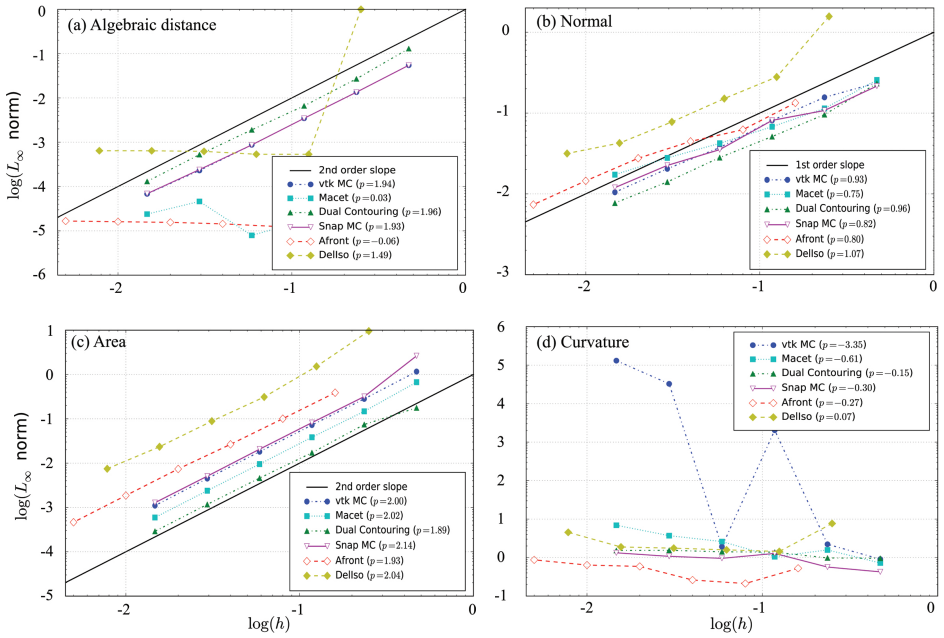


Figure 2.6. Observed order of accuracy after fixing Macet and Dual Contouring code (other curves remain the same). The black continuous line represents the expected behavior. p is the slope of the linear regression for each curve.

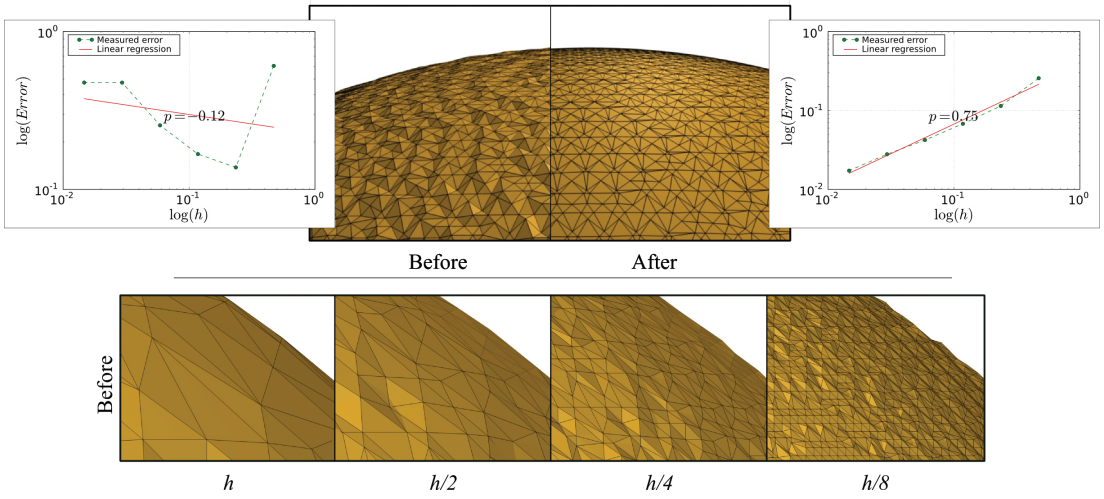


Figure 2.7. Through the verification methodology presented on this paper we were able to uncover a convergence problem within a publicly available marching-based isosurfacing code (top left) and fix it (top right). The problem causes the mesh normals to *disagree* with the known gradient field when refining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

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

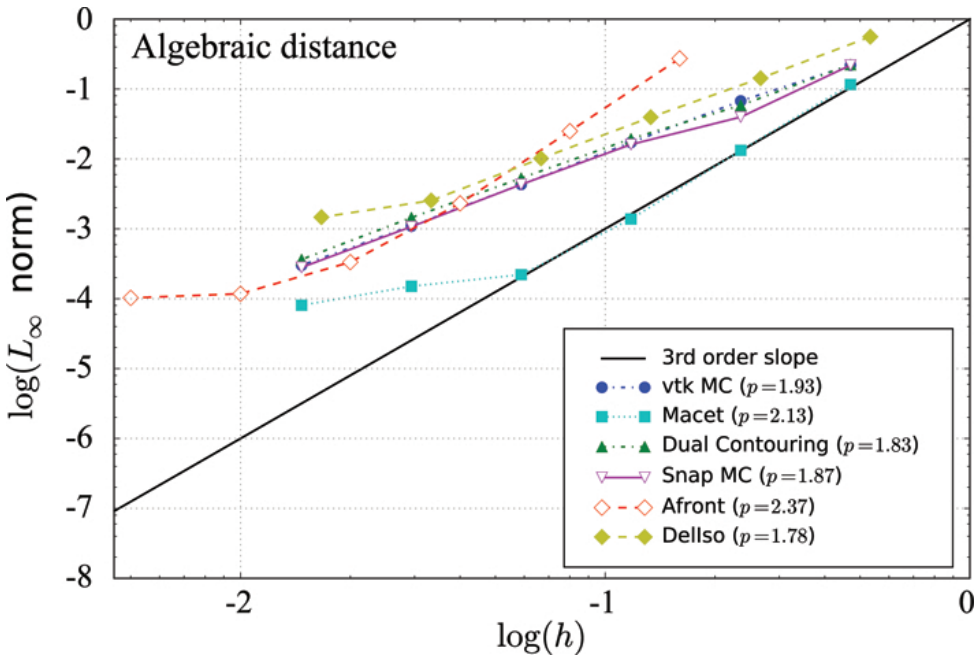


Figure 2.8. Order of accuracy for a transcendental function $f(x, y, z) = x^2 + y^2 + z^2 + \cos(Ax)^2 + \cos(Ay)^2 + \cos(Az)^2$, A is a constant. The observed orders of accuracy for all implementations are relative to the voxel size h. We expect third-order accuracy for Afront and Macet due to their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

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 complicated if curvatures are not constant over the surface (in that case, its additional parameter η comes into play [101], 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[47]. 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 [50]. 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 [31]. 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 [31], 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 [25, 101, 94], to performance [108] and accuracy analysis [86, 127].

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 [65] (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 [82]. Disambiguation mechanisms can ensure crack-free surfaces, and many schemes have been proposed, such as the one by Montani et al. [74], domain tetrahedralization [12], preferred polarity [5], gradient-based method [119], and feature-based schemes [43]. The survey of Newman and Yi has a comprehensive account [80]. 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 [81]. Nielson and Hamann, for example, use saddle points of the bilinear interpolant on grid cell faces [82]. 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 [78] and Chernyaev [17], 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 [18]. Subsequent work by Lopes and Brodlie [63] and Lewiner et al. [60] 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 [13], and Carr and Max [11]. 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 [36, 50], concrete proposals on verification are relatively recent. Etiene et al. [31] 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 [20, 26]. 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 [65]

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 [80]. 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 [82], 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 [17, 63, 81], 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

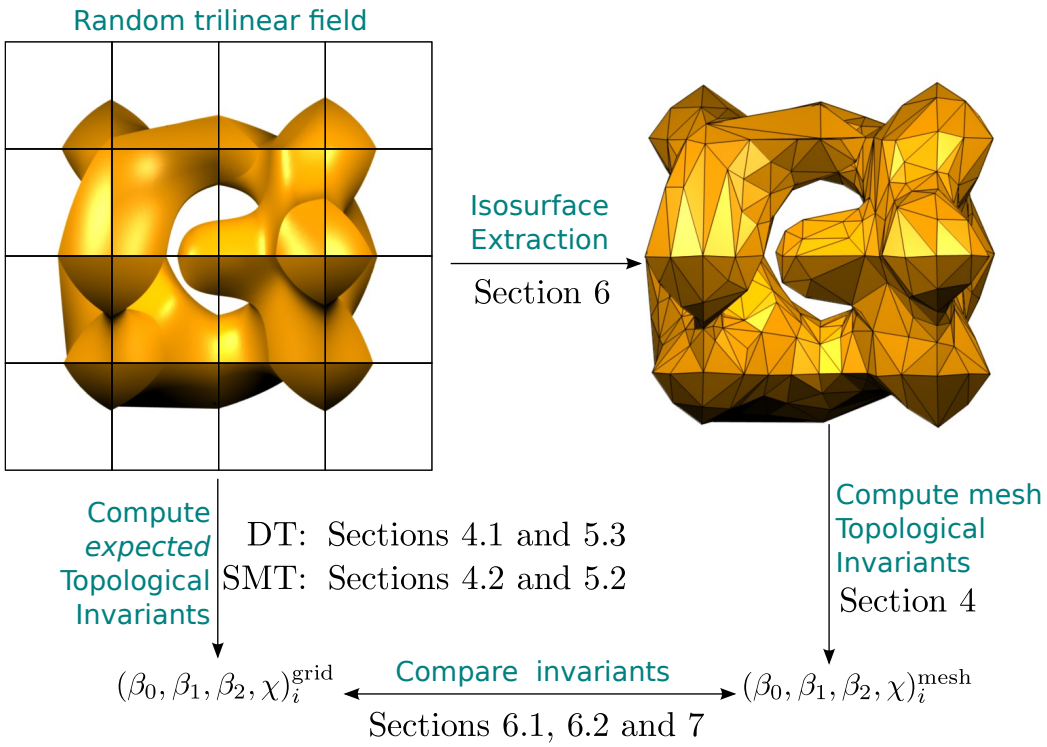


Figure 3.1. Overview of our topology verification pipeline. First step, we generate a random trilinear field and extract a random isosurface using the implementation under verification. We then compute the *expected* topological invariants from the trilinear field and compare them against the invariants 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).

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 [107], and Goresky and MacPherson give a comprehensive presentation of stratified Morse theory [37].

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 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 [26]. 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:

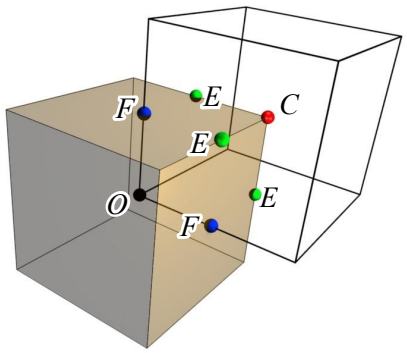


Figure 3.2. The four distinct groups of vertices O, F, E, C, are depicted as black, blue, green, and red points. They are the "Old", "Face", "Edge," and "Corner" points of a voxel region $V_{\mathcal{G}}$ (semitransparent cube) respectively. For the sake of clarity, we only show a few points.

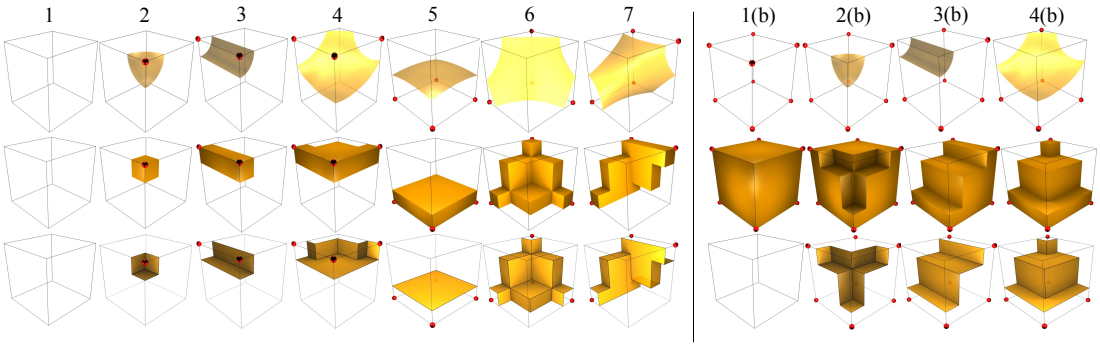


Figure 3.3. An illustration of the relation between unambiguous isosurfaces of trilinear interpolants and the corresponding digital surfaces. The top row shows all possible configurations of the intersection of $t = \alpha$ with a cube c_j for unambiguous configurations [63]. Each red dot s_i denotes 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).

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 [119]. If either property fails to hold, c_i is called *ambiguous*. The top row in Figure 3.3 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 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 3.2. 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 3.2 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 $e(v) \leq \alpha$
- $v \in F$ 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 3.3 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. [107]). Second, the genus of $\partial\mathcal{O}_\alpha$ can be computed directly from $\mathcal{O}_\alpha$ using the algorithm proposed by Chen and Rong [16] (Algorithm 3). 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 $t(x) = \alpha$ 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 [96, 107]. 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 3.3 [63], either a topological disk or the empty set. Each case in the top row of Figure 3.3 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 3.3) 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 3.3, respectively. Finally, we apply the Pasting Lemma [76] 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.

Algorithm 2 Voxel selection using Majority Interpolation (MI).

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

▷ Let O, F, E and C be the subset of vertices in $\mathcal{G}'$ as described in subsection 3.3.1.
▷ Let $\mathcal{N}(s, \star)$ be the set of neighbors of $s \in \mathcal{G}'$ in the set $\star$, where $\star = \{O, F, E, C\}$, with associate scalar less than α
for $s \in \mathcal{G}'$
    do if $s \in O$ **or**
        $s \in$ F and $|\mathcal{N}(s, O)| = 2$ **or**
        $s \in E$ and $|\mathcal{N}(s, O) + \mathcal{N}(s, F)| \geqslant 4$ **or**
        $s \in$ C and $|\mathcal{N}(s, O) + \mathcal{N}(s, F) + \mathcal{N}(s, E)| \geqslant 12$
            then Select voxel v_s
return $\mathcal{O}_\alpha$

Algorithm 3 A simple formula for genus computation.

GenusFromDS($\partial\mathcal{O}_\alpha$)

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

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 [37].

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

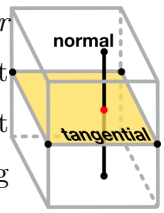
The above formulation is straightforward but unfortunately cannot be directly applied to functions appearing in either piecewise trilinear interpolations or isosurfaces with boundary, both of which appear in some of the isosurfacing algorithms with guaranteed topology. Trilinear interpolants are not 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 [121], 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 [13].

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 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*



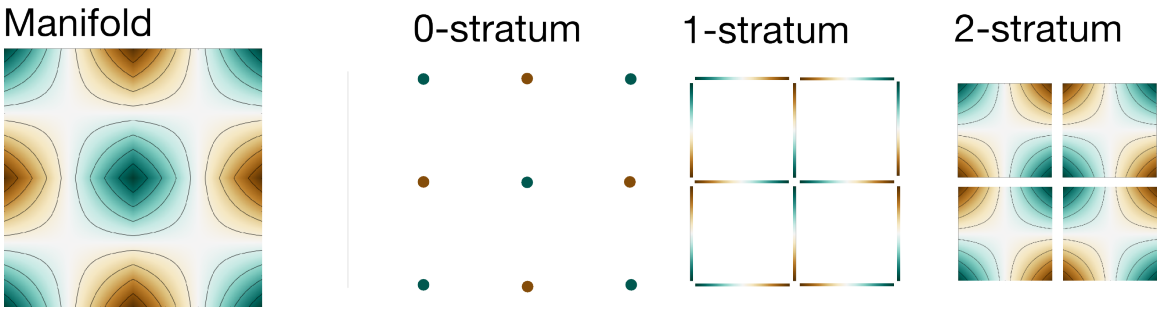


Figure 3.4. An illustration of a piecewise-smooth immersed 2-manifold. The colormap illustrates the value of each point of the scalar field. Notice that although the manifold itself is not everywhere differentiable, each stratum is itself an open manifold that is differentiable.

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 [81], by Carr and Snoeyink [13], and by Carr and Max [11]. 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 IsosurfacingRefers 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 4. 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:

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

Algorithm 4 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($\mathcal{G}$)

```
 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
```

where $\Delta_x = bc - ae$, $\Delta_y = bd - af$, and $\Delta_z = cd - ag$ [85]. 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.

3.4.3 Verification using Digital Topology

Algorithm 5 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 5 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 [81]. 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 3. Surfaces with boundary are avoided by assigning the scalar value 1 to every vertex

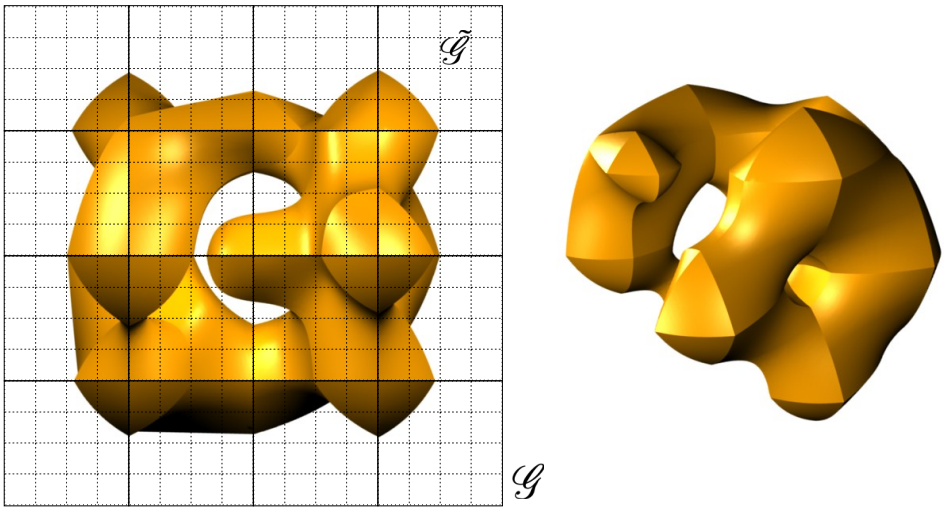


Figure 3.5. Our manufactured solution is given by $t(x) = \alpha$. $\mathcal{G}$ is depicted in solid lines while $\tilde{\mathcal{G}}$ is shown in dashed lines. $\tilde{\mathcal{G}}$ is a uniform subdivision of $\mathcal{G}$. The trilinear surfaces t_i are defined for each cube $c_i \in \mathcal{G}$ 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).

in the boundary of $\mathcal{G}$.

Algorithm 5 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 $\mathcal{G}$ be a $n \times n \times n$ rectilinear grid
2 **for** $i \leftarrow 1$ **to** #tests
3 **do** $\mathcal{G} \leftarrow$ randomly sampled $n \times n \times n$ grid
4 $\tilde{\mathcal{G}} \leftarrow$ RefineAndResample($\mathcal{G}$)
5 **if** $\tilde{\mathcal{G}}$ has ambiguous cubes
6 **then** GoTo 3
7 $\mathcal{O} \leftarrow$ MajorityInterpolation($\tilde{\mathcal{G}}$)
8 $K \leftarrow$ Isosurfacing($\mathcal{G}$)
9 $(\beta_0^v, \beta_1^v, \beta_2^v)_i \leftarrow$ InvariantFromDS($\partial\mathcal{O}$)
10 $(\beta_0^k, \beta_1^k, \beta_2^k)_i \leftarrow$ InvariantFromMesh(K)
11 Compare $(\beta_0^v, \beta_1^v, \beta_2^v)_i$ and $(\beta_0^k, \beta_1^k, \beta_2^k)_i$

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 [102] is the Visualization Toolkit (VTK) implementation of the Marching Cubes algorithm with the implicit disambiguation scheme proposed by Montani et al. [74]. 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 [25] 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 [101] 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® [68] 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 [94] 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 authors report an improvement in the minimum triangle angle when compared to previous techniques.

MC33 was introduced by Chernyaev [17] 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 [82] 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 [60] of Chernyaev's algorithm.

DELISO [24] 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. [99]. 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 4 and 5 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 [63, 81]. 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 [59]. 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.

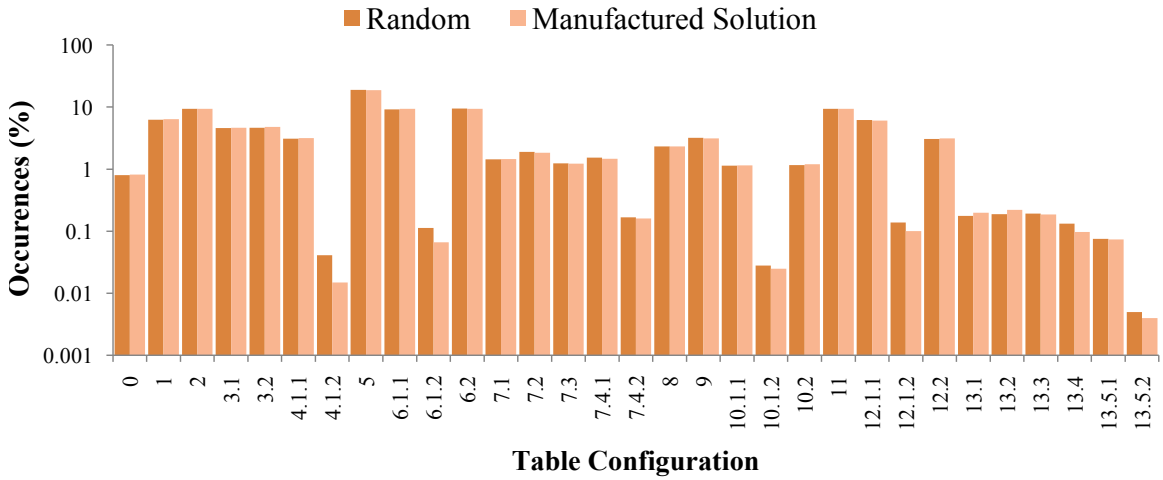


Figure 3.6. The horizontal axis shows the case and subcase numbers for each of the 31 Marching Cubes configurations described by Lopes and Brodlie [63]. The dark bars show the percentage of random fields that fit a particular configuration. The light bars show the percentage of random fields which fit a particular configuration *and* do not violate the assumptions of our manufactured solution. Our manufactured solution hits all possible cube configurations.

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 cubes of different sizes are glued together. MCFlow geometrical convergence is presented in the supplementary material [99].

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 [27] (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.

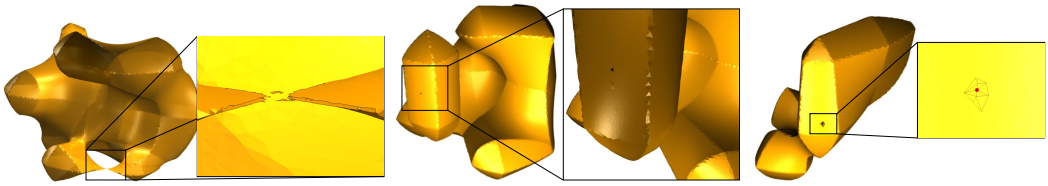


Figure 3.7. DelIso mismatch example. From left to right: holes in C^0 regions; single missing triangle in a smooth region; duplicated vertex (the mesh around the duplicated vertex is shown). These behavior induce topology mismatches between the generated mesh and the expected topology.

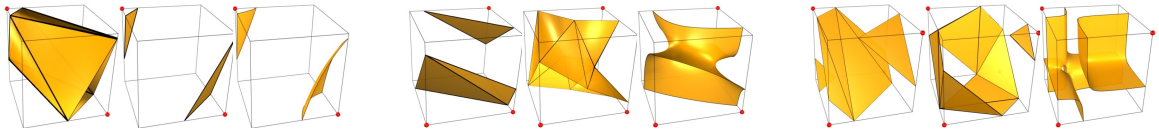
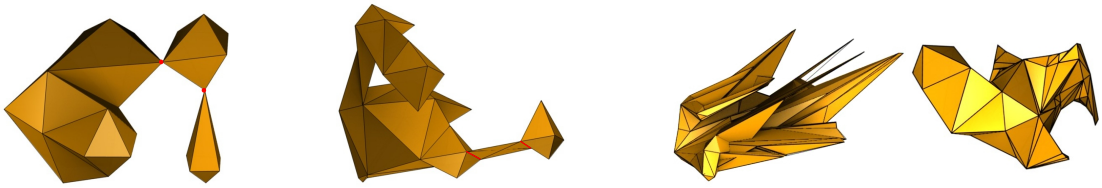


Figure 3.8. MC33 mismatch example. From left to right: problem in the case 4.1.2, 6.1.2, and 13.5.2 of marching cube table (all are ambiguous). Each group of three pictures shows the obtained, expected, and implicit surfaces. Our verification procedure can detect the topological differences between the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



(a) SnapMC (snap = 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.

	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

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.

3.6.2 Topology and Geometry

This work extends the work by Etiene et al. [31] 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 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. [107]. 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*[70].

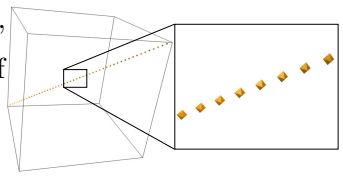
3.6.5 Contour Trees

Contour trees [14] 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[85]. 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 $d(x) = \alpha$ 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 [17, 85]. 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. [31] 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 MARCHING CUBES 33 TOPOLOGICAL CORRECTNESS

CHAPTER 5

VERIFYING DIRECT VOLUME RENDERING ALGORITHM

CHAPTER 6

RELIABLE VISUALIZATIONS

Text

6.1 Introduction

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 6.2 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[106] 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 6.3 we show how perception and user studies may impact flow

visualization, and in particular, we focus on issues related to color maps. In Section 6.4, 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 6.5, we speculate on some of the opportunities for collaboration and more effective communication between the two communities, and we conclude in Section 6.6.

6.2 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.

6.2.1 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[52]
		Hybrid	Color-coding and arrows[51]
		3D	Arrows in 3D space, 2-manifolds embedded in 3D[88]
		Enhancements	Large data[88], resampling[57]
	Color coding	Standard	Color maps, volume rendering[29]
Geometry	*Curve*	Streamline	Turk and Banks[117]
		Seeding	User-assisted[45], automatic[72, 62], and hierarchical[46]
		3D	2-manifolds embedded in 3D[105]
		Rendering	Illuminated[69], streamtubes and streamribbon[118]
		Unsteady	Wiebel and Scheuermann[124]
	Surface	Stream surface	Hultquist[44]
		Enhancements	Seeding and placement[87], accuracy[34]
		Unsteady	Schafhitzel *et al.*[98]
Texture	*LIC*	Standard	Cabral and Leedom[10]
		Performance	Improved algorithm, parallelism, real-time, GPU[61]
		3D	3D and 2-manifolds embedded in 3D[84]
		Rendering	Flow orientation cues, local velocity magnitude
		Unsteady	Li *et al.*[61]
	Spot Noise	Standard	van Wijk[120]
		Enhanced	It deals with highly curved/high velocity vector fields.[23]
		Performance	Parallel implementation.[58]
Feature	*VFT**	Standard	First-/High-order critical point tracking[40, 22, 100]
		Compression	Theisel *et al.*[110]
		Simplification	Weinkauf *et al.*[123]
		Streakline	Weinkauf and Theisel *et al.*[122]
	*STD***	Pathline	Theisel *et al.*[112]
	*LM****	FLTE	Haller[38], Garth *et al.*[33]

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

6.2.2 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.*[39] and Peng and Laramee[89] for an overview of the flow visualization field, Edmunds *et al.*[28] and McLoughlin *et al.* [71] for geometric flow visualization, Laramee *et al.*[56, 55] for texture-based flow visualization, and Pobitzer *et al.*[91] for feature-based flow visualization. Next, we briefly go over each of the classes (see Figure 6.1).

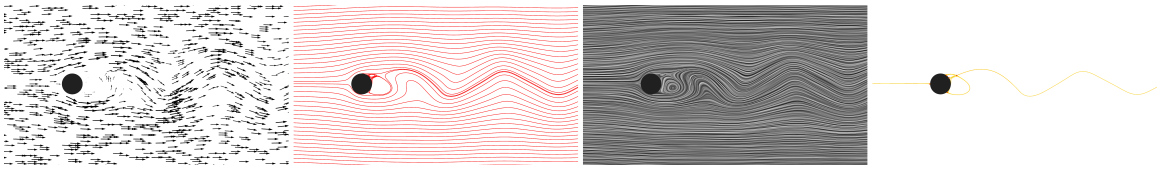


Figure 6.1. Examples of flow visualization using direct, geometry, texture-, and feature-based techniques, respectively.

6.2.2.0.1 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[29] 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.

6.2.2.0.2 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 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[28]. For curves, the main contributions of the past decade are related to rendering, seeding and placement of curves. Edmunds *et al.*[28] 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.

6.2.2.0.3 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[40] (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[111]. 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[38] 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[91]. 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.*[91].

6.2.2.0.4 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[28, 55].

6.2.3 Means to an end

In his position paper "On the death of visualization"[64], 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[118], which simultaneously encode the streamlines path and local flow vorticity, or from stream tubes[118], 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[57], 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[9]); 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.

6.3 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[115], 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.

6.3.1 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.[7, 66, 104].

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

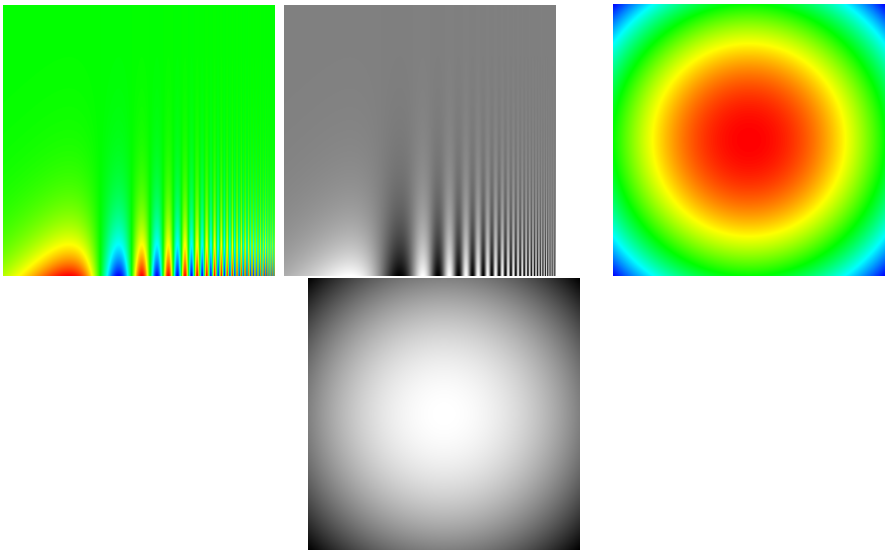


Figure 6.2. Left: the images show the color mapping of the spatial contrast sensitivity function. Frequency increases from left to right whereas contrast increases from the top to the bottom. The isoluminance of the rainbow color map obfuscate low contrast regions and small details, which can be seen 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.

map is ordered from shorter to longer wavelength of light, users do not easily perceive it as such, which makes quantitative analysis more difficult[7]. In addition, the rainbow 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[116]. 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[116]. Despite its disadvantages, the rainbow color map is widely used in the sciences. In the study by Borkin *et al.*[6], 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"[75]. 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 the rainbow color map[7]. 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.* [116], Moreland[75], Kindlmann *et al.*[49], and others[66, 113]. 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.*

6.3.2 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.*[54] comparing techniques for the visualization of steady 2D vector fields. The authors recruited five experts 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

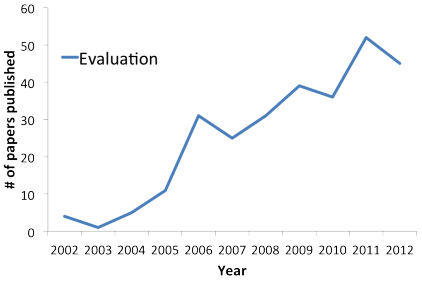


Figure 6.4. Evolution of the number of papers published on the topic of evaluation at TVCG.

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%

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.

(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 integral curves (with monoscopic and stereoscopic viewing) for 3D vector field data[32]. 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.

6.4 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.

6.4.1 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.*[97], 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.

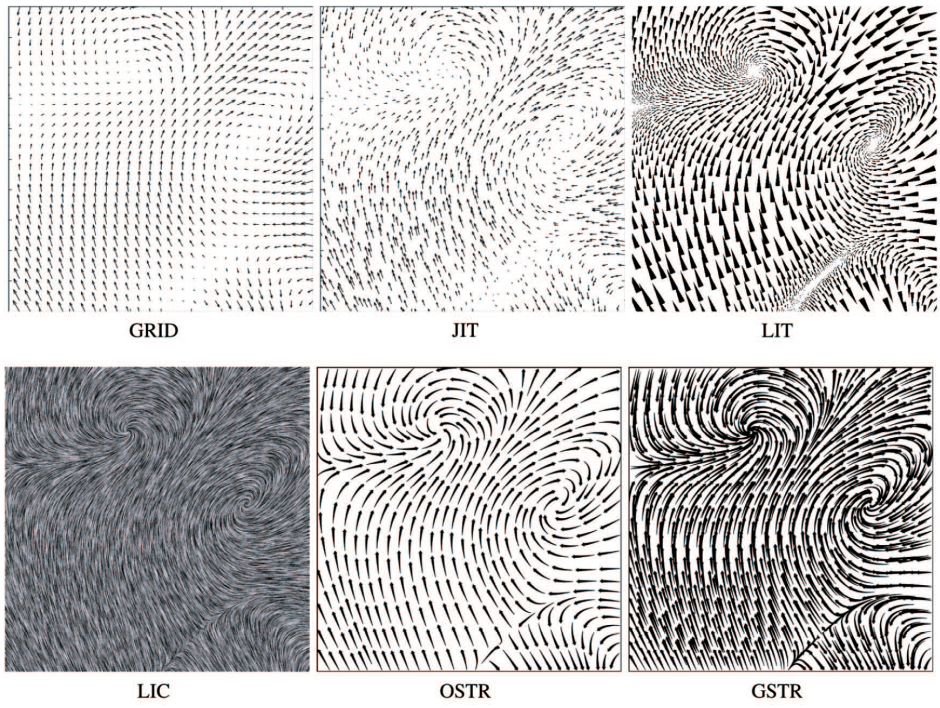


Figure 6.5. Comparing visualization methods for steady 2D vector fields. Top: standard arrow visualization, jittered arrow, icons using concepts borrowed from oil painting, respectively. Bottom: line-integral convolution, image-guided streamlines, streamlines seeded in a regular grid, respectively.

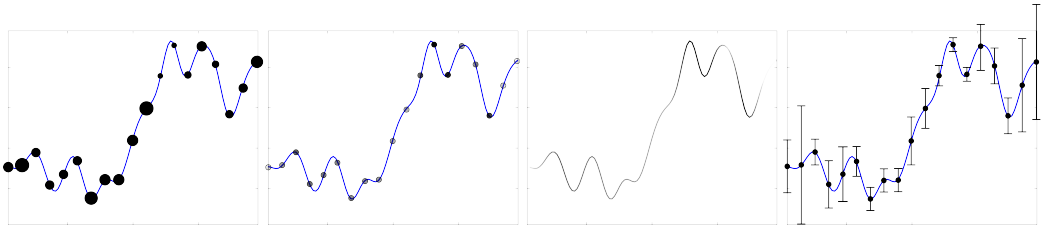


Figure 6.6. The four uncertainty visualization methods used by Sanyal *et al.*[97] in their user study. From left to right: glyphs size, glyphs color mapping, surface color mapping, and error bars.

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 their study. Nevertheless, a similar pattern can be found in the AIAA community (*e.g.*, see Figures 4 and 6 in Chassaing and Lucor[15]).

Several techniques for uncertainty visualization of vector fields are available. Botchen *et al.*[8] introduces a texture-mapping approach for uncertainty visualization of 2D vector fields. Hlawatsch *et al.*[42] 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.*[90] uses Gaussian random fields and takes into account spatial correlation of the data, which affects vector field features. Fout and Ma[79] 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.

6.4.2 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.*[31] 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 [30]. 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[37] and Digital Topology[107] 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[17], 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 [63]. 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.

6.5 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[77]. 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 [103].

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.

6.6 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

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] 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.

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

[6] 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).

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

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

[9] 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.

[10] 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.

[11] Carr, H., and Max, N. Subdivision analysis of the trilinear interpolant. *IEEE Computer Aided Geometric Design 16*, 4 (2010).

[12] 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.

[13] 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.

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

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

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

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

[18] 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.

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

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

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

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

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

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

[39] 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).

[40] 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.

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

[42] 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.

[43] 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.

[44] 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.

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

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

[47] Johnson, C. R., and Sanderson, A. R. A next step: Visualizing errors and uncertainty. *IEEE Computer Graphics and Applications 23*, 5 (2003), 6–10.

[48] Ju, T., Losasso, F., Schaefer, S., and Warren, J. Dual contouring of hermite data. In *SIGGRAPH'02* (2002), ACM, pp. 339–346.

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

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

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

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

[60] 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.

[61] 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.

[62] 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.

[63] 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.

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

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

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

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

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

[69] 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.

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

[71] 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.

[72] 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.

[73] 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.

[74] 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.

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

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

[77] 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.

[78] Natarajan, B. K. On generating topologically consistent isosurfaces from uniform samples. *Vis. Comput. 11*, 1 (1994), 52–62.

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

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

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

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

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

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

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

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

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

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

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

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

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

[97] 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.

[98] 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.

[99] 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.

[100] 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.

[101] 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.

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

[103] 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.

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

[105] 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.

[106] Squillacote, A. *The ParaView Guide: A Parallel Visualization Application*. Kitware, 2007.

[107] 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.

[108] 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.

[109] 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.

[110] 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.

[111] 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.

[112] 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.

[113] Tominski, C., Fuchs, G., and Schumann. Task-Driven Color Coding. In *Information Visualisation, 2008. IV '08. 12th International Conference* (2008), pp. 373–380.

[114] Tory, M., and Moeller, T. Human factors in visualization research. *IEEE Transactions on Visualization and Computer Aided Geometric Design Graphics 10*, 1 (2004), 72–84.

[115] Tory, M., and Möller, T. Human factors in visualization research. *IEEE Transactions on Visualization and Computer Graphics 10*, 1 (Jan. 2004), 72–84.

[116] Treinish, L., et al. Why should engineers and scientists be worried about color? *IBM Thomas J. Watson Research Center, Yorktown Heights, NY* (2009).

[117] 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.

[118] 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.

[119] van Gelder, A., and Wilhelms, J. Topological considerations in isosurface generation. *ACM Transactions on Graphics 13*, 4 (1994), 337–375.

[120] Van Wijk, J. Spot noise texture synthesis for data visualization. *ACM SIGGRAPH Computer Graphics* (1991).

[121] 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.

[122] 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.

[123] 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.

[124] Wiebel, A., and Scheuermann, G. Eyelet particle tracing-steady visualization of unsteady flow. In *Visualization, 2005. VIS 05. IEEE* (2005), IEEE, pp. 607–614.

[125] Xu, G. Convergence analysis of a discretization scheme for gaussian curvature over triangular surfaces. *Computer Aided Geometric Design 23*, 2 (2006), 193–207.

[126] 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.

[127] 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.