Visualization of Adaptive Mesh Refinement Data and Topology-based Exploration of Volume Data

Vom Fachbereich Informatik
der Universität Kaiserslautern
zur Erlangung des akademischen Grades

„Doktor der Naturwissenschaften“
(Dr. rer. nat.)

genehmigte Dissertation

von
Dipl.-Inform. Gunther H. Weber

Kaiserslautern
26. Mai 2003

1. Berichterstatter: Prof. Dr. Hans Hagen
2. Berichterstatter: Prof. Dr. Bernd Hamann
3. Berichterstatter: Hochschuldozent Dr. Gerik Scheuermann
Vorsitzender der Promotionskommission: Prof. Dr. Klaus Madlener
Dekan: Prof. Dr. Hans Hagen

Datum der wissenschaftlichen Aussprache: 16. September 2003

Meinen Eltern Ingrid Marie-Luise Weber† und Heinz August Weber und meinem Bruder Gernot August Weber...

Acknowledgments

> Twenty years from now you will be more disappointed by the things you didn't do than by the ones you did do. So throw off the bowline. Sail away from the safe harbor. Catch the trade winds in your sails. Explore. Dream. Discover.
>
> — *Mark Twain*

I want to use this opportunity to thank all people who made this work possible. First, I would like to thank my advisors Bernd Hamann, Gerik Scheuermann, and Hans Hagen. Bernd and Hans gave me the opportunity to conduct my research at two world-renowned research groups: the AG Graphische Datenverarbeitung und Computergeometrie, University of Kaiserslautern and the Visualization group at the Center for Image Processing and Integrated Computing, University of California at Davis. The continuous exchange proved to be a fruitful ground for developing new ideas. Gerik spawned my work on the analysis of scalar topology of trilinear interpolation. I value the countless discussions with him, which, apart from being enlightening, were very enjoyable.

I would also like to thank the members of the Computer Graphics Group in Kaiserslautern and the Visualization Groups at the University of California at Davis and the Lawrence Berkeley National Laboratory for their support of my work. In particular, I would like to thank Martin Öhler who helped substantially in the development of the parallel framework for distributed rendering of AMR data. I also thank Oliver Kreylos who contributed valuable ideas to my work. Terry Ligocki and John Shalf served as mentors, collaborators and supervisors at the Lawrence Berkeley National Laboratory (LBL). I do express my gratitude to both of them. During my time at LBL, Terry was much more than a "boss." Terry and John taught me a lot about AMR data. Furthermore, I would like to thank Kenneth I. Joy and Kwan-Liu Ma, who also contributed many ideas to my work about AMR data.

I would also like to thank my father, who supported me both mentally and financially during my studies, and my brother for being there for me. Finally, I would like to thank all my friends for keeping me sane during times of frustration and providing pleasant distraction from my work.

Zusammenfassung

Diese Dissertation behandelt zwei Hauptthemen: Visualisierung von Adaptive Mesh Refinement Simulationen und topologiebasierte Visualisierung von Skalarfeldern. Adaptive Mesh Refinement ist eine Simulationstechnik die 1984 von Berger und Oliger eingeführt wurde. Seitdem wird diese Technik zunehmend in physikalischen und ingenieurwissenschaftlichen Simulationen eingesetzt. Adaptive Mesh Refinement repräsentiert den Definitionsbereich einer Simulation als eine Menge regulärer Gitter bestehend aus Rechteckzellen. Diese Gitter sind in Hierarchieebenen mit wachsender Auflösung eingeteilt. Zur Zeit existieren nur wenige Visualisierungsalgorithmen, die die hierarchische Struktur von AMR Simulationen zur effizienten Visualisierung ausnutzen. Diese Dissertation erweitert einige Visualisierungstechniken mit dem Ziel, diese Struktur zu nutzen. Ich entwickle eine Methode, um Isoflächen ohne "Risse" (Diskontinuitäten) an Übergängen zwischen Hierchieebenen aus AMR Daten zu extrahieren. Daraufhin betrachte ich Volume-Rendering von AMR Daten. Indem ich eine AMR Hierarchie homogenisiere, das heißt, in Blöcke konstanter Aufösung aufteile, entwickle ich einen Ansatz, der eine AMR Hierarchie interaktiv darstellt. Zur Erzeugung hoch-qualitativer Darstellungen entwickle ich einen Cell-Projection-basierten Ansatz, der es erlaubt, Volume-Rendering Bilder einer Hierarchie fortschreitend zu verfeinern, indem ein Bild hierarchiestufenweise erzeugt wird. Basierend auf meinen Resultaten bei der Isoflächenextraktion entwickle ich eine C^0-stetige Interpolationsmethode, die zur Generierung hochqualitativer Bilder verwendet wird. Zum Abschluss meiner Abhandlung von AMR Daten entwickle ich ein Framework zum verteilten Rendern von AMR Daten auf Supercomputern und PC Clustern. Dabei entwickle und vergleiche ich unterschiedliche Verteilungsstrategien.

Im zweiten Teil dieser Arbeit stelle ich Methoden zur topologiebasierten Visualisierung von Skalarfeldern vor. Untersucht man ein Skalarfeld mit Hilfe von Isoflächen, so ist es oft schwierig Isowerte zu identifizieren, für die aussagekräftige Isoflächen entstehen. Mit Hilfe der Morsetheorie ist es möglich, kritische Punkte und zugehörige Werte zu identifizieren. An kritischen Punkten ändert sich die Topologie einer Isofläche: neue Flächenkomponenten entstehen, Flächenkomponenten verschmelzen, oder das Geschlecht einer Flächenkomponente ändert sich (Löcher entstehen oder verschwinden). Daher ist es möglich, "interessantes" Isoflächenverhalten aufgrund kritischer Punkte aufzufinden. Ich entwickle ein Verfahren, um kritische Werte für Skalardaten auf regulären Rechteckgittern mit stückweise trilinearer Interpolation zu erkennen. Außerdem stelle ich einen Ansatz vor, mit dem Benutzer das Verhalten von Isoflächen mit Hilfe kritischer Punkte untersuchen können. Des Weiteren benutze ich die kritischen Isowerte zur automatischen Generierung von Transferfunktionen für Volume-Rendering Anwendungen. Zum Abschluss meiner Arbeit erweitere ich das Konzept von kritischen Punkten auf kritische Regionen. Dies erlaubt es, Regionen mit konstantem Wert zu klassifizieren und meine Methode für eine größere Bandbreite von Datensätzen zu verwenden.

Summary

This dissertation deals with two main topics: visualization of adaptive mesh refinement (AMR) data and topology-based methods for visualization of volumetric data sets. AMR was introduced in the computational physics community by Berger and Oliger in 1984 and has gained increasing popularity in the computational physics community. AMR represents a domain as a set of structured, rectilinear grids at increasing resolution. Despite its inherent hierarchical nature, only few visualization algorithms exist that directly use the AMR structure for visualization purposes. In this dissertation I adapt several visualization techniques to make use of that hierarchical nature. I present an isosurface extraction scheme that allows for the extraction of crack-free isosurfaces from AMR data while representing each grid at the appropriate resolution. I subsequently consider volume rendering of AMR data. By homogenizing an AMR hierarchy, *i.e.*, partitioning it into blocks of constant resolution, I define a scheme that renders an AMR hierarchy at interactive speeds. To obtain high-quality volume-rendered visualizations, I develop a cell-projection-based approach that allows progressive rendering of AMR hierarchies by rendering an image level-by-level. Based on my work on crack-free isosurfaces, I develop a C^0-continuous interpolation scheme, which I use to generate high-quality images. I conclude my work on AMR data by developing a framework for parallel rendering of AMR data on supercomputers and PC clusters. I develop and compare different distribution strategies.

The second part of this dissertation introduces methods for topology-based exploration of scalar fields. When examining a scalar field using isosurfaces, it is often difficult to identify isovalues where relevant isosurface behavior occurs. Using Morse theory, it is possible to identify critical isovalues. Critical isovalues indicate isosurface topology changes: the creation of new surface components, merging of surface components or the formation of holes in a surface component. Therefore, they highlight "interesting" isosurface behavior and are helpful in exploration of large trivariate data sets. I present a method that detects critical isovalues in a scalar field defined by piecewise trilinear interpolation over a rectilinear grid. I then use the resulting list of critical points to aid users in examining a data set with isosurfaces. I further use critical isovalues to automatically generate transfer functions for direct volume rendering. I conclude my work by extending the concept of critical points to critical regions. This allows me to classify regions of constant value and use my method for a wider variety of data sets.

FB Informatik, Universität Kaiserslautern

Contents

Chapter 1

Introduction

Visualization is a crucial part in science and engineering. Since the amount of data collected in simulations and experiments has grown (and continues to grow) exponentially, powerful methods for understanding resulting data and gaining insight into underlying natural phenomena are also of increasing importance. In this dissertation, I deal with the visualization of volumetric scalar data. This type of data arises in a wide variety of application areas. In medical imaging, sampled data from computerized tomography (CT) or magnetic resonance imaging (MRI) scan provides density values within a volume. Simulations in computational fluid dynamics (CFD) compute scalar values like pressure or density at locations in a volume.

The work on this dissertation started with the visualization of AMR data. AMR is a simulation technique that was introduced in the computational physics community by Berger and Oliger in 1984 [7]. A modified technique was introduced by Berger and Colella in 1989 [6]. Since then, the popularity of this technique has grown. Today, it is commonly used in numerical simulations of a wide variety of scientific and engineering phenomena or processes. Adaptive mesh refinement aims at combining the simplicity and efficiency of regular, rectilinear grids with the ability to adapt to changes in resolution normally inherent to unstructured meshes. By representing a domain as a set of regular, rectilinear grids at increasing resolution, AMR combines the efficiency of simulations based on regular grids with the ability to adapt to changes in resolution inherent in simulations based on unstructured meshes. Regions that cannot be represented with sufficient accuracy by a coarse grid are refined by a set of higher-resolution grids. Recently, AMR has been used for efficiently simulating astrophysical phenomena [8] that span several orders of magnitudes and push AMR to its limits.

The inherent hierarchical nature of AMR data lends itself to be used in visualization applications. However, when the work on this dissertation started, only few visualization algorithms existed that used this structure. The first goal was to visualize AMR data using direct volume rendering and isosurfaces. I started by developing a cell-projection-based approach for direct volume rendering of AMR data using constant interpolation. The goal was to support progressive rendering of AMR hierarchies. A user should be able to start rendering a coarse level of an AMR hierarchy. Subsequently, contributions from higher resolution level should be incorporated in the resulting image while providing the ability to view intermediate results. I then used the piecewise linear method for interpolation within cells to improve image quality. The use of the

piecewise linear method was motivated by the fact that it was one of the interpolation methods used during simulation and that it worked on the original grid. It soon became obvious that a pure software-based approach was too slow for choosing view points and transfer functions. By "homogenizing" an AMR data set, *i.e.*, partitioning it into blocks of constant resolution, it was possible to render individual blocks using an arbitrary hardware-accelerated volume rendering scheme. I implemented a new scheme that simulated my cell-projection approach in hardware at a lower quality. I aimed to use the AMR hierarchy to achieve interactive frame rates by rendering only parts of an AMR hierarchy.

I then considered the extraction of isosurfaces. AMR data is commonly given in cell-centered format while visualization methods like marching cubes (MC) expect vertex-centered data. Resampling AMR grids to vertex centered format leads to "dangling" nodes and results in cracks in an extracted isosurface. Rather than fixing the cracks, which are caused by resampling, I decided to avoid the resampling step completely. The fundamental idea was to use the original samples in an isosurface extraction scheme. Within single grids of an AMR hierarchy, this is possible by considering the locations of the samples as vertices of a grid that is "dual" to the original grid. Using dual grids leads to gaps between hierarchy levels. By developing a procedural scheme to fill these gaps with "stitch cells" and extending the case table of marching cubes, I was able to extract crack-free isosurfaces from AMR data while fully considering the hierarchy and automatically reacting to changes in resolution. The piecewise-linear method used during the simulation is not continuous. This leads to artifacts in volume rendered images. By defining interpolants for all cells produced by my grid stitching scheme, I was able to use dual grids and stitch cells to produce high-quality volume rendered images.

In certain instances, for example while debugging, constant interpolation is well suited. I developed a framework that uses an efficient re-implementation of the cell-projection volume renderer to render AMR hierarchies on supercomputers and PC clusters. I implemented and compared several distribution strategies. The aim was to provide a framework for testing distribution strategies and determine whether PC clusters are a viable alternative to hardware-accelerated methods.

The second part of this dissertation introduces methods for topology-based exploration of scalar fields. When examining a scalar field using isosurfaces, it is often difficult to identify isovalues where relevant isosurface behavior occurs. Topological methods have proved their value in vector field visualization and initial work on topology-based methods for scalar fields looked promising. Topology studies properties of a surface that do not change under deformation, like number of components and the genus (*i.e.*, the number of holes). Morse theory provides ways to detect critical points where the topology of an isosurface changes. When considering large data sets, critical isovalues with their associated topology changes are certainly of interest to a user. The goal was to detect these topology changes for isosurfaces extracted by marching cubes. The criteria for critical points in Morse theory are based on the first and second derivative of an analytical function. However, trilinear interpolation is only C^0-continuous, thus preventing the use of the original criteria from Morse theory. By considering alternate definitions based on geometric properties of critical points, used by other researchers to detect critical points for linear interpolation on tetrahedral meshes, I was able to detect critical points for piecewise trilinear interpolation. By using a variant of MC that preserves that topology I am able to provide a user

with information on a MC isosurface. It is possible to zoom in on critical points and examine topological changes in detail. It turned out that it is also possible to use the resulting list of critical isovalues for automatic transfer function design.

Morse theory defines critical points. However, in real data sets, regions often exist that assume a constant value. In this case, it is necessary to classify whole regions. For example, a torus can form around a polyline constituting a local minimum. I extended the concept of critical points to critical regions and developed an approach to classify critical regions. This enables me to handle a larger variety of data sets. The contributions of this dissertation are:

- A hardware-accelerated volume rendering method for previewing AMR data [80]
- A scheme for extracting crack-free isosurfaces from AMR data [81, 82].
- A progressive, high-quality, cell-projection-based volume rendering scheme for AMR data [80] that utilizes stitch cells to define a consistent interpolation scheme [83].
- A framework for testing distribution strategies for parallel rendering of AMR data, and the comparison of various distribution strategies [84].
- A scheme to detect critical isovalues for piecewise trilinear interpolation [85].
- The generalization of this scheme to the detection of critical regions [86].

The remainder of this dissertation is structured as follows: Chapter 2 describes the theoretical foundations of my work. It gives a short introduction to topology, and it describes different data formats for volumetric scalar data. Subsequently, I examine foundations of volume rendering and isosurface extraction. An emphasis is put on the description of the marching cubes method, as my work on topological analysis of scalar data depends on a thorough understanding of the behavior of this standard visualization algorithm. Chapter 2 concludes with a description of the format of Berger-Collela AMR data. Chapter 3 describes previous work on visualization of AMR data and topology-based methods for the visualization of scalar data. Chapters 4 and 5 deal with visualization of AMR data. Chapter 4 describes my crack-free isosurface extraction method, and Chapter 5 various volume rendering schemes for AMR data. Chapter 6 introduces my methods for detecting critical points and regions for scalar fields and presents ways to use them when visualizing volumetric data. Chapter 7 provides conclusions and suggests possible avenues for future work.

Chapter 2

Theoretical Foundations

2.1 Topology

2.1.1 Introduction

Topology studies properties of a shape that do not change under deformation [34, 35, 87]. Formally, topology is defined via the concept of *open sets* which are defined as follows:

Definition 1 (Topological Space) *A pair $(X, \mathcal{O})$ consisting of a set X and a collection $\mathcal{O}$ of subsets (called open sets) of X is a topological space if the following axioms hold for $\mathcal{O}$:*

A1) Arbitrary unions of open sets, i.e., sets in $\mathcal{O}$, are open, i.e., in $\mathcal{O}$.

A2) The intersection of two open sets is open.

A3) The (trivial) subsets $\varnothing$ and X are open.

$\mathcal{O}$ is the topology of the topological space $(X, \mathcal{O})$.

Usually, one omits the topology in notation and simply refers to X as the topological space. Informally, topology deals with connectedness. The open sets of X define the concepts of neighborhood and connectedness in X. This fact is illustrated using metric spaces as an example.

Definition 2 (Metric Space) *A metric space is a pair (X, d) consisting of a set X and a function $d : X \times X \to \mathbb{R}$, i.e., the metric, with the following properties*

M1) $d(x, y) \geq 0$ for all $x, y \in X$, and $d(x, y) = 0$ if and only if $x = y$.

M2) $d(x, y) = d(y, x)$ for all $x, y \in X$.

M3) $d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in X$ ("Triangle Inequality").

Definition 3 (Topology of a Metric Space) *Let (X, d) be a metric space. A subset $V \subset X$ is open if for every $x \in V$ there exists an $\epsilon > 0$ such that the "closed ϵ-ball" $\mathcal{D}_\epsilon(x) := \{y \in X | d(x, y) \leq \epsilon\}$ around x lies completely within V. The topology of the metric space (X, d) is the set $\mathcal{O}(d)$ of all these open sets of X.*

Property M3 of the metric allows us to conclude that for each location y with $d(x, y) < \epsilon$ there exists a smaller closed δ-ball that is completely contained in the closed ϵ-ball around x. This is analogous to the situation in $\mathbb{R}^n$ with the Euclidean norm. In a metric space, the open sets are all sets that contains an ϵ-ball.

A function f is continuous if it preserves topology, *i.e.*, open sets, leading to the following definition

Definition 4 (Continuous Function) *Let X and Y be topological spaces. A function $f : X \rightarrow Y$ is continuous if the preimage of an open set in Y is an open set in X.*

For metric spaces this definition is equivalent to the commonly known definition from analysis: "For each $\epsilon > 0$ there exists a $\delta > 0$ ..."

Homeomorphisms are one way to draw an equivalence between two shapes that share equal connectivity. Two objects A and B are *homeomorphic* (topologically equivalent) if there is a mapping (*i.e.*, a function) f that transforms (deforms) A into B satisfying the following requirements:

H1) F is *onto* (surjective), *i.e.*, for each location on B there exists a location on A that is transformed to it.

H2) F is *one-to-one* (injective), *i.e.*, each point on A maps to a unique point on B

H3) F is *bicontinuous*, *i.e.*, F is continuous in both "directions."

Properties H1 and H2 ensure that there is a unique correspondence between each point on A and each point on B. No "overlap" exists. Property $H3$ ensures that one cannot tear A, join sections of A together, poke holes into A, or seal up holes in A. For example, a torus is topologically equivalent to a coffee cup. A torus can be deformed into a cup by pulling one end of the torus out and forming it into a cup. (The hole in the torus becomes the handle of the cup.) A sphere and a torus, on the other hand, are not topologically equivalent. It is not possible to deform a sphere into a torus without tearing it since it does not have a hole like the torus.

In computer graphics and visualization objects are commonly represented as surfaces. From a topology viewpoint, these surfaces are 2-manifolds, *i.e.*, they are locally euclidean and topologically equivalent to a disk. In general, a d-manifold is locally homeomorphic to the open unit ball in $\mathbb{R}^d$. A property holds locally when for each point on the manifold a neighborhood, *i.e.*, a connected set of points with a maximum distance of ϵ, exists for which the property holds. The open unit ball in $\mathbb{R}^d$, indicated as $\mathcal{U}^d$, is a ball of radius one, and is defined as

$$\mathcal{U}^d := \left\{ \mathbf{x} \in \mathbb{R}^d \middle| \|x\| < 1 \right\} \ . \tag{2.1}$$

Another way to define topological equivalence is homotopy equivalence. Differing from a homeomorphism, homotopy equivalence disregards the number of dimensions. Two shapes are homotopy equivalent if they have the same number of components and holes. For example, a circle and a solid torus are homotopy equivalent (the circle is a deformation retract of the torus [34]) even though a circle is a 1-manifold and a solid torus is a 3-manifold.

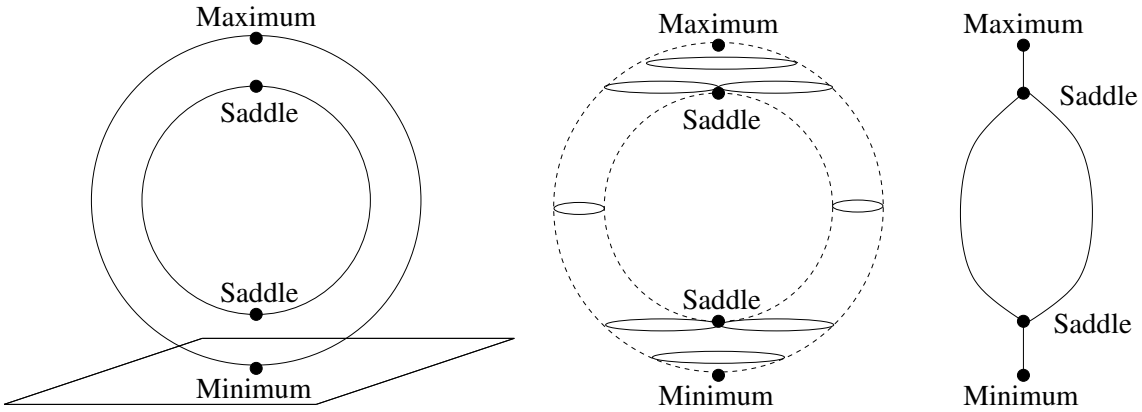


Figure 2.1: Classical example illustrating Morse theory. (a) An "upright" torus, with a height function defined as distance from the base plane, has four critical points. (b) At each critical point the topology of the contour obtained by intersecting the torus with a horizontal plane changes. (c) A Reeb graph encodes critical points and corresponding topological changes.

2.1.2 Morse Theory

Morse theory [34, 64] relates global topological properties of a manifold to differential critical points of a smooth real-valued function f. Since values from this function are interpreted as height, it is often referred to as *height function*. Consider a d-manifold $\mathbb{M}$ and a differentiable real-valued function f. By definition, $\mathbb{M}$ is locally topologically equivalent to a open unit ball in $\mathbb{R}^d$ and one can choose a local orthonormal system of coordinates x_i, $1 \leq i \leq n$. At a *critical point* $\mathbf{x}$ on the manifold, the gradient of f with respect to a local coordinate system vanishes, *i.e.*,

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}(\mathbf{x}), \frac{\partial f}{\partial x_2}(\mathbf{x}), \ldots, \frac{\partial f}{\partial x_d}(\mathbf{x}) \right) = 0 \; . \tag{2.2}$$

The critical point type can be determined from the signs of the eigenvalues of the *Hessian*

$$H(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 x_d}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_d^2}(\mathbf{x}) \end{pmatrix} \; . \tag{2.3}$$

Morse theory only considers isolated *non-degenerate* critical points. A critical point $\mathbf{x}$ is non-degenerate if all eigenvalues of the Hessian $H(\mathbf{x})$ differ from zero. A non-degenerate critical point $\mathbf{x}$ can be classified based on the signs of the eigenvalues of the Hessian $H(\mathbf{x})$. When all eigenvalues are positive $\mathbf{x}$ is a minimum. When all eigenvalues are negative $\mathbf{x}$ is a maximum. Otherwise, $\mathbf{x}$ is a saddle.

The classical example for Morse theory is considering the surface of a torus standing on its side, see Figure 2.1(a). The real-valued height function of a point on the torus is defined as the

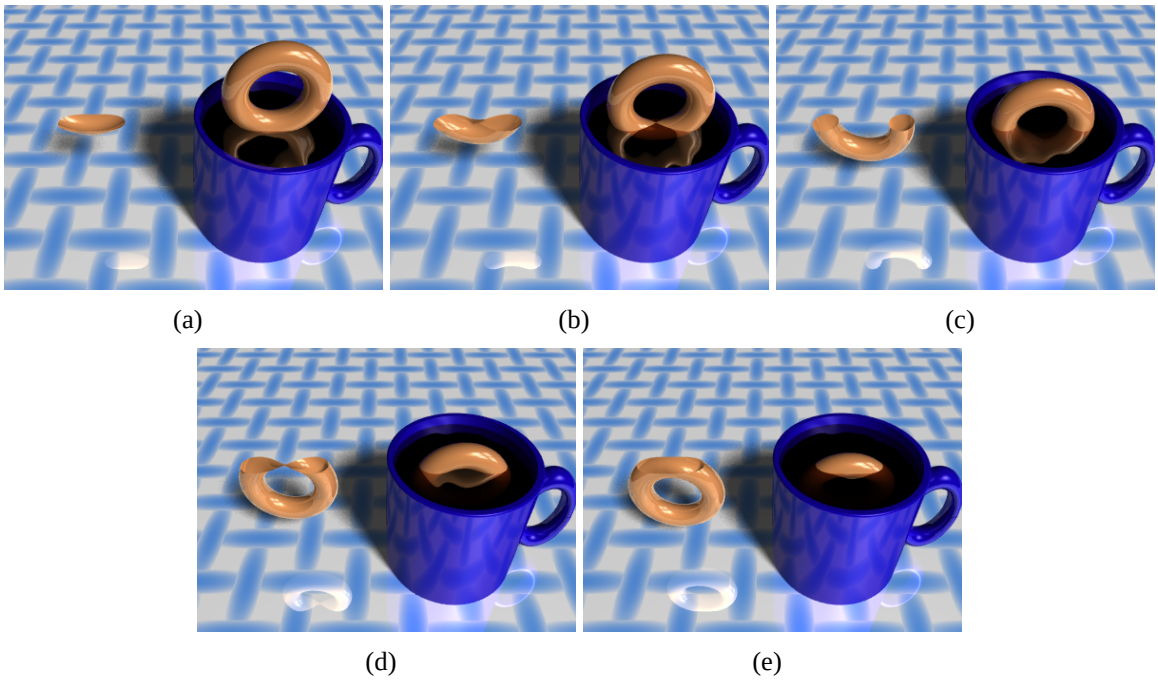


Figure 2.2: Components of a torus below a clipping plane. (Images courtesy of John C. Hart)

measured distance from the "base plane." This function implies four critical points: a minimum at the lowest point of the torus; two saddles at the lowest and highest points of the inner ring; a maximum at the highest point of the torus. Morse theory examines the topology of a contour at a certain height. The contour is the intersection of the shape with a plane consisting of all points $\mathbf{p}$ that have equal height h with respect to the height function, *i.e.*, all points where $f(\mathbf{p}) = h$ holds. The fundamental statement of Morse theory is that the topology of this contour only changes at heights of critical points, see Figure 2.1(b). For the torus in the example this contour changes as follows: at the minimum, a contour appears; at the first saddle, this contour splits into two separate components; at the second saddle these components merge; at the maximum, the contour vanishes.

A *Reeb graph*, see Figure 2.1(c), encodes topological changes of contours at a certain height and is obtained by associating two points $\mathbf{p}$ and $\mathbf{q}$ on the shape when they belong to the same component of the contour at a certain height. Edges in the resulting graph correspond to connected components on a cross section at a certain height. Nodes in this graph correspond to critical points. A Reeb graph is the *topological skeleton* of a surface.

When the surface of the torus is clipped with a plane at a certain height, the topology of the "clipped" component changes only at the critical points. Hart [34] illustrates these changes by "dunking" a torus-shaped donut into a cup of coffee and considering the donut component below the coffee surface, see Figure 2.2. Initially, the clipped torus is the "empty set." There is no component below the clip plane. At the minimum, a new component that is topologically equivalent to a disk appears, see Figure 2.2(a). At the first saddle this component changes into a

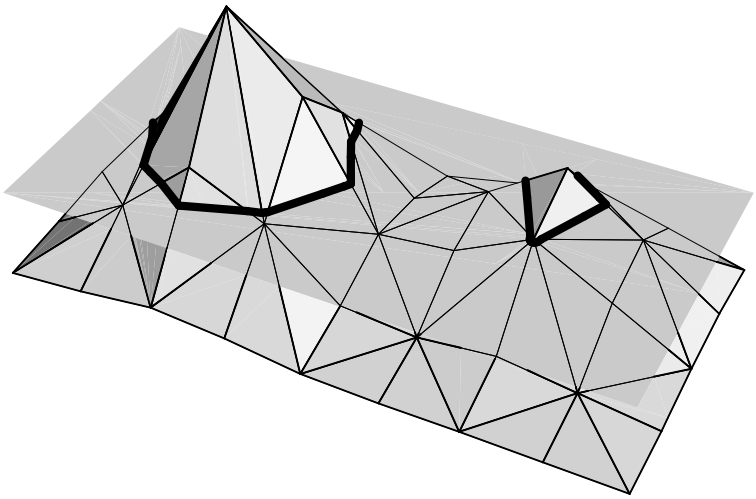


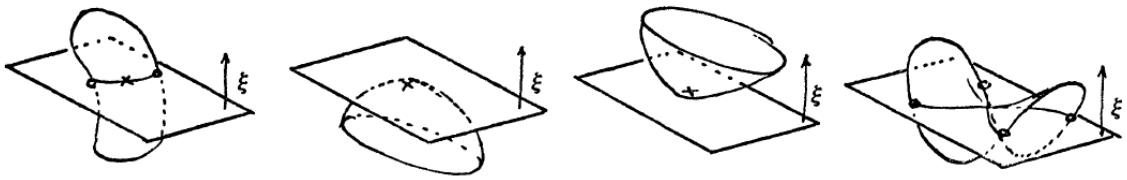
Figure 2.3: A terrain data set associates each location in a domain with a scalar value, interpreted as height. Contours, *i.e.*, curves connecting points of equal height, are a commonly used visualization technique. (Image courtesy of Kraus and Ertl [46])

cylinder, see Figures 2.2(b) and 2.2(c). At the second saddle the clipped component changes into a "pierced torus," see Figures 2.2(d) and 2.2(e). At the maximum the punctured torus is closed.

For the torus, the choice of a height function is arbitrary. Different choices for height functions lead to different configurations of critical points and different Reeb-graphs. (Imagine, for example, a lying torus instead of a torus standing on its side.) There are surfaces for which a unique choice of a height function exists. A 2-d terrain data set, for example, is defined via a function that assigns an elevation to each location of a 2-d domain, see Figure 2.3. Thus, the obvious choice for a height function is the function defining the terrain itself. For terrain data, contours are a well-known visualization technique and are commonly used in geographical maps. Again, contour topology changes only at critical points, *i.e.*, points where the gradient vanishes. The type of topology change depends on the type of critical point, determined by the sings of the eigenvalues of the Hessian: at a minimum, a new component appears; at a maximum a component vanishes; at a saddle components either merge or separate.

Morse theory can be generalized to higher dimensions. A three-dimensional (3-d) scalar field (or volume data set) can be treated as height field in 4-d space defined on a 3-d domain. Contours correspond to isosurfaces which are a common visualization technique for volumetric data, see Section 2.2.5. It is possible to determine isosurface topology based on the relationship of the isovalue (*i.e.*, height value) with respect to values of critical points. Contour behavior at minima and maxima is equivalent to the 2-d case. Contour behavior at saddles can differ. In addition to merging and separating with other contour components, a contour component can change its genus at a saddle, *i.e.*, a hole in a surface component can appear or vanish.

Banchoff [4] observed that while Morse theory defines critical points via differential properties they are also defined by inherent geometric properties. His approach considers a height function ξ which is obtained by choosing a vector $\boldsymbol{\xi}$ and projecting points of the surface onto



(a) (b) (c) (d)

Figure 2.4: Classifying a point $\mathbf{p}$ on a smooth surface by considering a "small circle" on the tangent plane. Point $\mathbf{p}$ is denoted by a small "x" in the image. (Images courtesy of Banchoff [4])

that vector. The height function for the example of the torus standing on its side on a plane is obtained by choosing the plane normal as $\boldsymbol{\xi}$. For a terrain model, given on a domain within the xy-plane, $\boldsymbol{\xi}$ is collinear with the z-axis.

For a smooth surface, it is possible to classify a point $\mathbf{p}$ concerning a height function ξ by considering a "small disc neighborhood" and a "small circle" around $\mathbf{p}$ on the surface and their relation to a horizontal plane, *i.e.*, a plane of equal height, through $\mathbf{p}$. The small disc neighborhood consists of all points on the surface whose distance to $\mathbf{p}$ is smaller than a fixed distance. The small circle is the boundary of the small disc neighborhood. At an ordinary (or regular) point, see Figure 2.4(a), the tangent plane at $\mathbf{p}$ is not horizontal. A horizontal plane through $\mathbf{p}$ intersects the small circle exactly twice and partitions the small disc neighborhood in exactly two components. At a critical point, the tangent plane is horizontal. At a minimum or a maximum, see Figures 2.4(b) and 2.4(c), the small circle does not meet the tangent plane at all. All points of the small disc neighborhood (with the exception of $\mathbf{p}$) lie below or above a horizontal plane. At a saddle, see Figure 2.4(d), the small circle intersects the horizontal plane through $\mathbf{p}$ more than two times (four times in the figure) and partitions the small disc neighborhood into more than two regions.

On a polyhedral surface, critical points can only occur at vertices of the shape. For a polyhedral surface, the small disc neighborhood of a point $\mathbf{q}$ can be defined as $\mathrm{Star}(\mathbf{q})$, *i.e.*, the set of all vertices, edges and faces which include $\mathbf{q}$. The small circle corresponds to the polygon boundary of $\mathrm{Star}(\mathbf{q})$. In analogy to the smooth case, vertices are classified based on the number of intersection points of the boundary polygon with a horizontal plane through $\mathbf{q}$.

Edelsbrunner *et al.* [19] used the criteria developed by Banchoff for piecewise linear functions defined on triangulated domains. By defining the *lower star* $\underline{\mathrm{Star}}(\mathbf{q})$ as the portion of $\mathrm{Star}(\mathbf{q})$ lying below (or on) the horizontal plane through $\mathbf{q}$ and the *upper star* $\overline{\mathrm{Star}}(\mathbf{q})$ as the portion of $\mathrm{Star}(\mathbf{q})$ lying above (or on) the horizontal plane through $\mathbf{q}$, it is possible to distinguish between critical point types based on the number of "wedges" (*i.e.*, connected components consisting of triangles) in $\underline{\mathrm{Star}}(\mathbf{q})$ and $\overline{\mathrm{Star}}(\mathbf{q})$. If $\overline{\mathrm{Star}}(\mathbf{q}) = \mathrm{Star}(\mathbf{q})$ then $\mathbf{q}$ is a minimum. If $\underline{\mathrm{Star}}(\mathbf{q}) = \mathrm{Star}(\mathbf{q})$ then $\mathbf{q}$ is a maximum. If neither $\underline{\mathrm{Star}}(\mathbf{q})$ nor $\overline{\mathrm{Star}}(\mathbf{q})$ equals $\mathrm{Star}(\mathbf{q})$, then $\mathbf{q}$ is a regular point if each consists of exactly one wedge. Otherwise $\mathbf{q}$ is saddle with a multiplicity based on the number of wedges in $\underline{\mathrm{Star}}(\mathbf{q})$ and $\overline{\mathrm{Star}}(\mathbf{q})$. Edelsbrunner *et al.* [18] subsequently extended their definitions to 3-d data on tetrahedral meshes.

(a) (b)

Figure 2.5: Structured versus unstructured domain. (a) In a structured domain, an inherent relationship between sample position, *e.g.*, samples are located at fixed distances with respect to another, exists. (b) In an unstructured domain, sample positions are arbitrary.

2.2 Foundations of Volume Visualization

2.2.1 Trivariate Data

A scalar field F is a function that assigns a scalar valued quantity, like temperature, or pressure, to each location in the domain $D \subset \mathbb{R}^3$, *i.e.*,

$$F : D \rightarrow \mathbb{R}.$$

In physics and mathematics, a scalar field is usually a function that is often a solution to a set of differential equations. In scientific visualization, data is commonly given as a set of samples. These samples can result from measurements, *e.g.*, medical CT or MRI scans, or are obtained by numerical simulation. In numerical simulation, equations describing a physical phenomenon are discretized and a numerical approximation to the analytical solution is computed.

Each sample s_i has a position $\mathbf{p}_i$ and an associated function value v_i. Values at arbitrary positions in the domain are obtained by *interpolation*. The sample domain can be classified by considering sample positions, see Figure 2.5. In a *structured* domain, an inherent relationship between sample positions exists. For example, samples are located within a fixed distance from each other, see Figure 2.5(a). In the structured case, sample positions can be given implicitly enabling very compact data storage. Only sample values and high-level information about positions, *e.g.*, spacing, need to be stored. Structured domains also allow for quickly locating of a sample, commonly in constant time. In unstructured domains, see Figure 2.5(b), sample positions are arbitrary. Locations must be stored explicitly along with sample values. Searching samples for a given location is computationally more expensive and is commonly implemented employing auxiliary space-partitioning data structures.

Data can be given in form of *scattered data*, where no further "connectivity" information, besides sample position, is given. (Foley and Hagen [22] surveyed methods used in scattered data interpolation.) Alternatively, data is given on a mesh, where a neighborhood relationship between samples is established by a grid consisting of cells that connect samples. Figure 2.6 shows commonly used cell types for 3-d meshes. The cuboid cell, see Figure 2.6(d), is a special case of the hexahedral cell, see Figure 2.6(e).

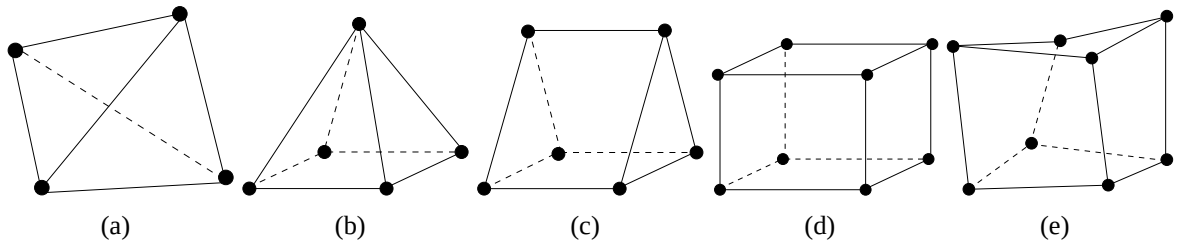


Figure 2.6: Cell types commonly used in meshes: tetrahedron (a), pyramid (b), triangle prism (c), cuboid (d), and hexahedron (e), *i.e.*, a more general form of a cuboid.

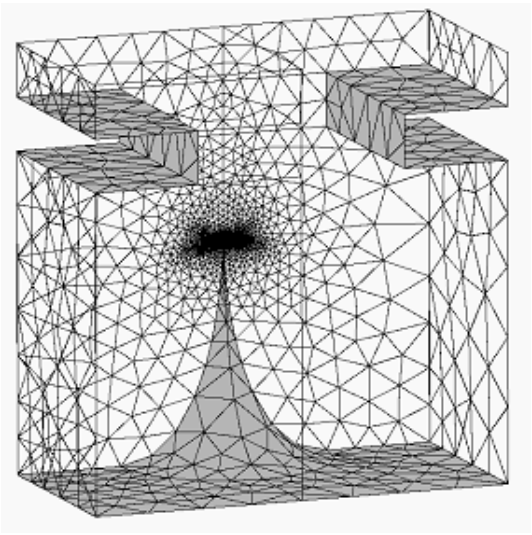


Figure 2.7: 3-d unstructured mesh. (Image courtesy of CFD Research Corporation http://www.cfdrc.com)

Meshes can be structured or unstructured. Unstructured meshes, see Figure 2.7, commonly use tetrahedral cells, but can also use any of the cell types shown in Figure 2.6. This grid representation requires more storage space than a structured grid, as both sample positions and grid structure need to be stored.

Structured meshes are composed of hexahedral cells. Assuming that samples are accessible via three indices (i, j, k), cells are given implicitly by connecting samples with adjacent indices in each direction. The domain can be either structured or unstructured. Structured meshes on a structured domain require the least storage since both sample positions and grid cells are given implicitly.

Interpolation over meshes is done by specifying an interpolant for each cell type. The value at an arbitrary position is interpolated by locating the cell that contains that position and then computing the interpolated value from the values associated with the vertices of that cell. Finding a cell for a given position is easy for structured meshes on a structured domain and can be usually

performed in constant time. In an unstructured mesh, it is more difficult to locate a cell that contains a given position.

2.2.2 Interpolation

Introduction

In a mesh, values are interpolated by locating the cell containing a point and then computing the interpolated value as a combination of the values at the vertices (and additional locations for higher-order cells). Distinguished positions in a cell, where values are given, are referred to as *nodes*, and associated values are called *nodal values*. An interpolation scheme for a cell is commonly given by specifying a set of *basis functions*. For each node, its corresponding basis function assigns a weight within the interval $[0, 1]$ to each location within the cell. This weight specifies the "contribution" of the nodal value to the interpolated value at that particular location. If C is a cell with n vertices, and $B_i(\mathbf{x})$ is the basis function belonging to node i having an associated nodal value value v_i, then an interpolated value $I(x)$ at a point $\mathbf{x}$ within the cell is computed as

$$I(\mathbf{x}) = \sum_{i=0}^{n} B_i(\mathbf{x}) v_i \ . \tag{2.4}$$

At each position $\mathbf{x}$ within a cell, all basis functions are positive and sum to one, *i.e.*, $\forall \mathbf{x} \in C : \sum_{i=0}^{n} B_i(\mathbf{x}) = 1$. Basis functions are often given for *standard elements*. The standard element for a cuboid cell, for example, is the unit cube. Interpolated values for arbitrary cells are computed as follows: First, coordinates of a point within the cell are transformed to coordinates within the standard element corresponding to that cell. Subsequently, an interpolated value is calculated within the standard element. Considering, for example, univariate linear interpolation, the standard element is the interval $[0, 1]$. Nodal values are v_0 at position 0 and v_1 at position 1. An interpolated value $I(x)$ for a position $x \in [0, 1]$ within the standard element is computed as

$$I(x) = (1 - x)v_0 + xv_1 \ , \tag{2.5}$$

using the basis functions

$$\begin{aligned} B_0(x) &= (1 - x) \ \text{ and} \\ B_1(x) &= x \ . \end{aligned} \tag{2.6}$$

Interpolating in arbitrary "cells," defined by nodes x_0 and x_1, is achieved by computing the interpolation coordinate t within the standard element, obtained as

$$t = \frac{x - x_0}{x_1 - x_0} \ . \tag{2.7}$$

The resulting coordinate is then used to compute an interpolated value using the standard element functions, *i.e.*,

$$I(t) = (1 - t)v_0 + tv_1. \tag{2.8}$$

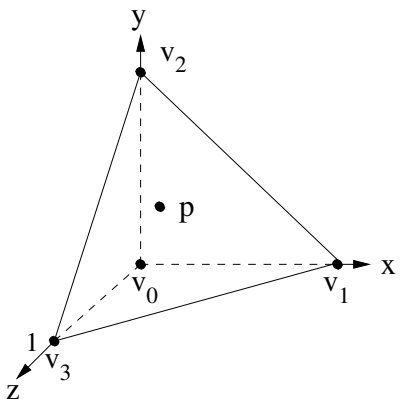


Figure 2.8: Standard element for a linear tetrahedron.

Linear Tetrahedron

In a linear tetrahedron, values are obtained by linear interpolation from nodal values. Figure 2.8 shows the standard element for a linear tetrahedron. The basis functions for this standard element are

$$\begin{aligned} B_0(x,y,z) =& 1-x-y-z \ , \\ B_1(x,y,z) =& x \ , \\ B_2(x,y,z) =& y \text{ , and} \\ B_3(x,y,z) =& z \ . \end{aligned} \tag{2.9}$$

Assuming that a nodal value v_i has an associated position $\mathbf{p}_i$, each position $\mathbf{p}$ within a tetrahedron can be expressed as *barycentric combination* of the node positions $\mathbf{p}_i$, defined as linear combination

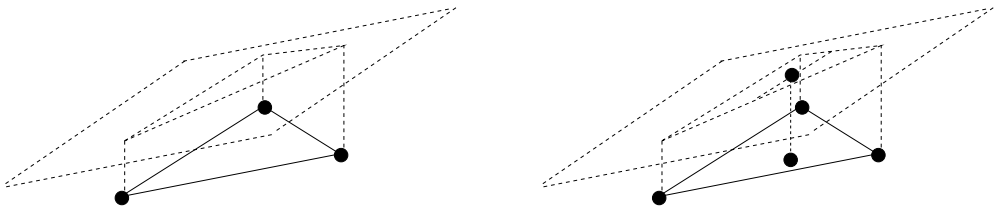
$$\mathbf{p} = \beta_0(x,y,z)\mathbf{p}_0 + \beta_1(x,y,z)\mathbf{p}_1 + \beta_2(x,y,z)\mathbf{p}_2 + \beta_3(x,y,z)\mathbf{p}_3 \ . \tag{2.10}$$

The β_i are the *barycentric coordinates* of $\mathbf{p}$ and are equivalent to the basis functions B_i. Within a tetrahedron, each barycentric coordinate $\beta_i(x,y,z)$ has a value between zero and one and the sum of all barycentric coordinates equals one. For arbitrary tetrahedra, the barycentric coordinate $\beta_i(x,y,z)$ of a position $\mathbf{p}$ can be computed as the fraction of the volume of the tetrahedron obtained by replacing $\mathbf{p}_i$ with $\mathbf{x}$ with respect to the volume of the complete tetrahedron defined by the $\mathbf{p}_i$, *e.g.*,

$$\beta_0 = \frac{V_{\mathbf{x}\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3}}{V_{\mathbf{x}\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3}} \ . \tag{2.11}$$

The volume of an arbitrary tetrahedron T is

$$\text{Vol}(T) = \frac{1}{6}\begin{vmatrix} x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix} \ , \tag{2.12}$$



(a) (b)

Figure 2.9: On boundary faces, linear interpolation within a tetrahedron reduces to linear interpolation within a triangle. Interpreted as "height," all interpolated values lie on a plane (a). Thus, values at an arbitrary position (denoted by a solid disc) within the triangle can be obtained by linear interpolation between the two edge values lying on a line passing through that position (b).

where x_i, y_i and z_i are the coordinates of the i-th vertex defining the tetrahedron. Since the determinant is linear with respect to each row, the β_i of the barycentric combination are linear with respect to the location of $\mathbf{x}$. As a result, an interpolated value is linearly dependent on $\mathbf{x}$ and can be expressed as

$$I(\mathbf{x}) = Ax_0 + Bx_1 + Cx_2 + D \ , \tag{2.13}$$

where A, B, C and D are real coefficients.

Interpolated values can be thought of lying on a 3-d-hyperplane. On boundary face triangles, interpolation reduces to linear interpolation within a triangle, see Figure 2.9(a). (Which can be defined analogously via a barycentric combination of points in a triangle. In fact, the same concept holds for arbitrary simplices.) Interpreted as "height," all interpolated values lie on the plane defined by the three nodal values. Along edges, linear interpolation is used. Because all interpolated values lie on a plane, values at an arbitrary position $\mathbf{p}$ within the triangle can be obtained by interpolating linearly between two edge values lying on a line passing through $\mathbf{p}$, see Figure 2.9(b). Within a tetrahedron, interpolated values can be obtained by linearly interpolating between values on boundary faces lying on an arbitrary line passing through $\mathbf{p}$. This property is commonly used by cell-based volume rendering schemes operating on tetrahedra. During cell-projection, for example, see Section 2.2.4, it is possible to interpolate values linearly when scan-converting boundary triangles of a tetrahedral cell. Values within the cell can be obtained by linearly interpolating between values at front- and back-face of a tetrahedron.

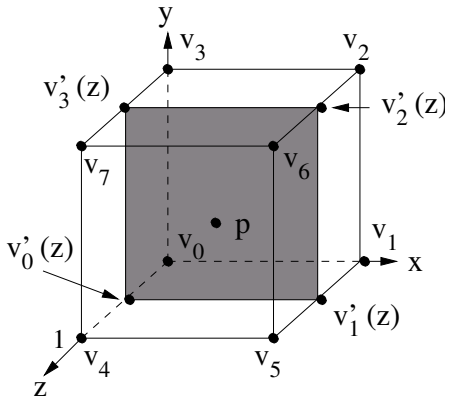


Figure 2.10: Standard element for trilinear cuboid with vertex numbering scheme.

Trilinear Cuboid

The standard element for the trilinear cuboid is the unit cube, see Figure 2.10. The basis functions for the trilinear cuboid are

$$\begin{aligned}
B_0(x,y,z) &= (1-x)(1-y)(1-z)\ , \\
B_1(x,y,z) &= x(1-y)(1-z)\ , \\
B_2(x,y,z) &= xy(1-z)\ , \\
B_3(x,y,z) &= (1-x)y(1-z)\ , \\
B_4(x,y,z) &= (1-x)(1-y)z\ , \\
B_5(x,y,z) &= x(1-y)z\ , \\
B_6(x,y,z) &= xyz\ \text{, and} \\
B_7(x,y,z) &= (1-x)yz\ .
\end{aligned}$$

Coordinates within an axis-aligned cuboid $\mathbf{p}' = (x', y', z')$, where $x' \in [x_0, x_1]$, $y' \in [y_0, y_1]$, $z' \in [z_0, z_1]$ can be mapped to the standard cube by the normalization $x = \frac{x'-x_0}{x_1-x_0}$, $y = \frac{y'-y_0}{y_1-y_0}$, $z = \frac{z'-z_0}{z_1-z_0}$. General hexahedral cells can be mapped to the unit cube using linear transformations.

Trilinear interpolation is equivalent to a series of subsequently applied linear interpolation steps. Starting in z-direction, values along each of the four z-parallel edges are obtained by linear interpolation. Subsequently, values on the two yz-planes are obtained by linearly interpolating in y-direction between the two values at the edges. (As a result, trilinear interpolation reduces to bilinear interpolation when restricted to a boundary face.) Finally, a value within the cube is obtained by linearly interpolating along a x-axis-aligned line between the two values on the yz-boundary planes. Any order in which these interpolations are performed yields the same result. Trilinear basis functions are *tensor product* functions of linear basis functions.

Differing from linear interpolation in a tetrahedron, trilinear interpolation reduces only to linear interpolation when axis-aligned directions are used. One cannot just linearly interpolate between two values at intersection points of a line with the cell's boundary faces, as in general

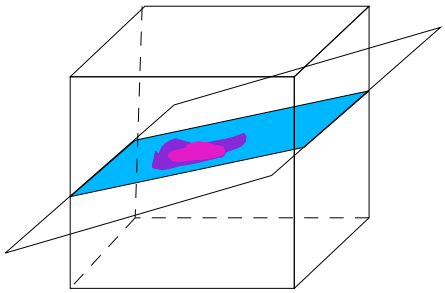


Figure 2.11: Visualization of scalar volume data using a cutting plane. Using a color map each point on the plane is colored according to the scalar value at that position.

this approach yields incorrect results. Correct results are only obtained when interpolating along axis-aligned lines.

2.2.3 Visualization Methods

Volumetric data is usually visualized using one of three fundamental visualization techniques:

Cutting Planes Using a *color map*, a color is assigned to each scalar value. The domain of the scalar field is intersected with a plane, and each point on that plane is displayed in the color associated with the scalar value at that location, see Figure 2.11. In the further course of this work, cutting planes are not considered.

Direct Volume Rendering Using a *transfer function*, optical properties (emission/color, absorption/opacity, *etc.*) are assigned to each scalar value. Subsequently, a viewpoint is chosen and a resulting image is rendered.

Isosurface extraction A surface connecting all locations where the scalar field assumes a certain *isovalue* is extracted, *i.e.*, $f^{-1}(\{v\})$ is computed and displayed.

2.2.4 Direct Volume Rendering

Introduction

Direct volume rendering visualizes volumetric scalar data sets by first assigning optical properties to scalar values using a *transfer function* or *color map* and rendering a resulting image from a given viewpoint. One can classify volume rendering methods by their underlying illumination models (*i.e.*, the "optical properties" of transfer functions) and by their operation in *image* or *object space*. Two illumination models are widely used in volume rendering: The *absorption and emission light model* described by Max [62] and the Phong-based light model introduced by Levoy [51].

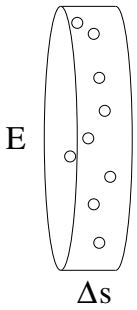


Figure 2.12: The absorption light model: A slab with a base area of E and a thickness Δs.

Image generation is performed in image or object space. Image-space algorithms, including the commonly used ray casting algorithm, see Sabella [73], operate on pixels in screen space as "computational units," *i.e.*, they perform computations on a per-pixel basis. Object-space methods, like cell projection, see Ma and Crockett [58], operate on 3-d grid cells. Another example for an object-space algorithm is splatting, see Westover [89]. Hybrid-approaches working in both image and object space exist. Volume rendering using the shear-warp factorization, see Lacroute *et al.* [49] and Lacroute [48], is the most prominent example.

Optical Models

Max [62] surveyed light-models used for direct volume rendering. The absorption and emission light model described in that paper is commonly used in scientific visualization. To understand the absorption and emission light model, one has to understand its components, the *absorption light model* and the *emission light model.* All models are described in terms of achromatic light using only luminance information. Color is added by sampling luminance for discrete spectral colors. Even though physically inaccurate, most volume rendering approaches specify three samples (a red, a green, and a blue component) for each color. They furthermore specify one achromatic opacity or transparency component, thus assuming that absorption is uniform for all colors/wavelengths.

The *absorption light model* assumes that a medium is filled with cold, perfectly black particles. The *particle density* ρ specifies the number of particles per unit volume in the medium. For simplicity, it is assumed that all particles are identical spheres with a radius of r. If one considers a small cylindrical slab with a base area E and length Δs, see Figure 2.12, this slab has a volume of $E\Delta s$ and contains $N = E\Delta s\rho$ particles. Each particle has a projected area of $A = \pi r^2$ on the cylinder base. If the slab length Δs is sufficiently small, it is unlikely that particle projections on the base overlap. Consequently, the area of the base overlapped by particles can be approximated as $NA = E\Delta s\rho A$. Thus, the fraction of occluded light $\frac{\Delta I}{I}$, which is equivalent to the overlapped fraction of the base area, is given as

$$\frac{\Delta I}{I} = -\frac{E\Delta s\rho A}{E} = -\Delta s\rho A\ . \tag{2.14}$$

Occlusion reduces the flux of light, indicated by the negative sign of ΔI in the equation. As Δs approaches zero, the probability for overlap of particle projections also approaches zero. The approximation of Equation 2.14 becomes exact when using the differential equation

$$\frac{dI}{I} = -ds\rho(s)A \ , \tag{2.15}$$

which can be rewritten as

$$\frac{dI}{ds} = -\rho(s)AI(s) = \epsilon(s)I(s) \ . \tag{2.16}$$

The solution to this differential equation,

$$I(s) = I_0 \exp\left(-\int_0^s \epsilon(t)dt\right) \ , \tag{2.17}$$

gives the intensity at parameter value s along the ray. I_0 is the "background intensity," *i.e.*, the intensity at $s = 0$. The "observer" is located at parameter value $D > 0$ along the ray and perceives an intensity of $I(D)$.

The emission light model assumes that particles in the medium are glowing and completely transparent. Particles only add light to a ray through the medium. Each particle glows with an intensity of C per unit projected area. Its projected area $E\Delta s\rho A$ contributes a glow flux of $CE\Delta s\rho A$ to the base area E, adding a flux of $C\rho A\Delta s$. As Δs approaches zero, the differential equation

$$\frac{dI}{ds} = C(s)\rho(s)A = C(s)\tau(s) = g(s) \tag{2.18}$$

results. The term $g(s) = C(s)\tau(s)$, called *source term*, specifies the added flux. By modifying this term, it is possible to include additional effect as, for example, reflection, see Max [62]. The solution to this differential equation is

$$I(s) = I_0 + \int_0^s g(t)dt \ . \tag{2.19}$$

"Real" particles occlude light as well as creating light. The corresponding differential equation is the combination of Equations 2.16 and 2.18 and is given as

$$\frac{dI}{ds} = g(s) - \tau(s)I(s) \ . \tag{2.20}$$

Max [62] derived the solution to this equation as

$$I(D) = I_0T'(D) + \int_0^D g(s)T'(s)ds \ , \tag{2.21}$$

where

$$T'(s) = \exp\left(-\int_s^D \tau(x)dx\right) \tag{2.22}$$

is the transparency between parameter value s and the "observer" located at parameter value D along the viewing ray.

The commonly used *particle model* is based on identical spherical particles and uses a source term $g(s) = C(s)\tau(s)$. In volume visualization, the glow intensity $C(s)$ and extinction coefficient $\tau(s)$ are chosen according to a scalar value at the position in the data set associated with the parameter s. This choice is based on a transfer function that maps scalar values to values for C and τ. Instead of specifying an extinction coefficient τ in the transfer function, it is common to specify and use an opacity value $\alpha(s)$. Opacity specifies the percentage of light that remains after a ray has passed through a layer of material of unit thickness having an extinction coefficient $\tau(s)$. Thus, this opacity value $\alpha(s)$ is defined as

$$\alpha(s) = 1 - \exp\big(-\tau(s)\big) \; . \tag{2.23}$$

The extinction coefficient can be obtained from the opacity by

$$\tau(s) = -\ln\big(1 - \alpha(s)\big) \; . \tag{2.24}$$

If one assumes constant glow (or color) C and opacity α (implying a constant extinction coefficient τ) along a ray segment (*i.e.*, a continuous part of the ray) spanning a parameter interval $[t_0, t_1]$, values for the integrals in Equation 2.22 and Equation 2.21 can be obtained analytically. Analogously to $T'(s)$, the transparency between the observer and a position s along the ray segment, T'_{Seg} is defined as transparency between segment start parameter t_0 and a position $s \in [t_0, t_1]$. This transparency can be obtained analytically using

$$\begin{aligned} T'_{\text{Seg}}(s) &= \exp\left(-\int_{t_0}^{s} \tau dk\right) = \exp\big(-\tau(s - t_0)\big) = \big(\exp(-\tau)\big)^{s-t_0} \\ &= \exp\big(\ln(1-\alpha)\big)^{s-t_0} = (1 - \alpha_{\text{Cell}})^{s-t_0} \end{aligned} \tag{2.25}$$

The opacity of the complete ray segment is calculated as

$$\alpha_{\text{Seg}} = 1 - T'_{\text{Seg}}(t_1) = 1 - (1-\alpha)^{t_1 - t_0} \; . \tag{2.26}$$

Using Equation 2.21, the color (intensity) of a ray segment can be computed as

$$\begin{aligned} C_{\text{Seg}} &= \int_{t_{\text{in}}}^{t_{\text{out}}} C\tau T'_{\text{Seg}}(s) ds = C\tau \int_{t_0}^{t_1} T'_{\text{Seg}}(s) ds \\ &= -C\ln(1-\alpha) \int_{t_0}^{t_1} (1-\alpha)^{s-t_0} ds = \frac{C\ln(1-\alpha)(1-\alpha)^{t_1-t_0)}}{\ln(1-\alpha)} \\ &= C\alpha_{\text{Seg}} \; . \end{aligned} \tag{2.27}$$

When combining contributions of two adjacent ray segments having constant colors C_0 and C_1 and constant opacities α_0 and α_1, the combined transparency is computed by

$$\begin{aligned} T_{\text{Combined}} &= \exp\left(\int_{t_0}^{t_2} \tau(k) dk\right) = \exp\left(\int_{t_0}^{t_1} \tau_0 dk + \int_{t_1}^{t_2} \tau_1 dk\right) \\ &= \exp\left(\int_{t_0}^{t_1} \tau_0 dk\right) \exp\left(\int_{t_1}^{t_2} \tau_1 dk\right) = T_{\text{Seg0}}(t_1) T_{\text{Seg1}}(t_2) \\ &= (1 - \alpha_{\text{Seg0}})(1 - \alpha_{\text{Seg1}}) \; . \end{aligned} \tag{2.28}$$

(One assumes that Seg0 ranges from t_0 to t_1 and Seg1 from t_1 to t_2.) The combined opacity of the segments is

$$\alpha_{\text{Combined}} = 1 - (1 - \alpha_{\text{Seg0}})(1 - \alpha_{\text{Seg1}}) = \alpha_{\text{Seg0}} + (1 - \alpha_{\text{Seg0}})\alpha_{\text{Seg1}} \ . \tag{2.29}$$

The combined color (intensity) is computed as

$$C_{\text{Combined}} = C_{\text{Seg0}} + (1 - \alpha_{\text{Seg0}})C_{\text{Seg1}} \ . \tag{2.30}$$

(This is equivalent to compositing pixels using colors with pre-multiplied alpha values, see Porter and Duff [72].)

If one considers arbitrary functions along ray segments, it may not be possible to evaluate the integrals in Equation 2.21 analytically, and it becomes necessary to use numerical approximations. The simplest numerical approximation for an integral is the Riemann sum, defined as

$$\int_0^D h(x)dx \approx \sum_{i=1}^{n} h(x_i)\Delta x \ .$$

The interval from zero to D is split into n segments of equal length $\Delta x = \frac{D}{n}$. Within each segment, a sample location $x_i \in [(i-1)\Delta x, i\Delta x]$ is chosen. The integral is approximated assuming that h has a constant value $h(x_i)$ within the complete interval. If one considers a whole ray and approximate all integrals in Equations 2.21 and 2.22 using the same sample positions, the Riemann sum is equivalent to splitting the ray into n segments and assuming constant color and opacity within each segment. An approximation for the transparency between the observer and a sample location is

$$T'_{approx}(s_i) = \prod_{j=0}^{i-1} \big(1 - \alpha_{\text{Sample}}(s_j)\big)^{\Delta x} \quad . \tag{2.31}$$

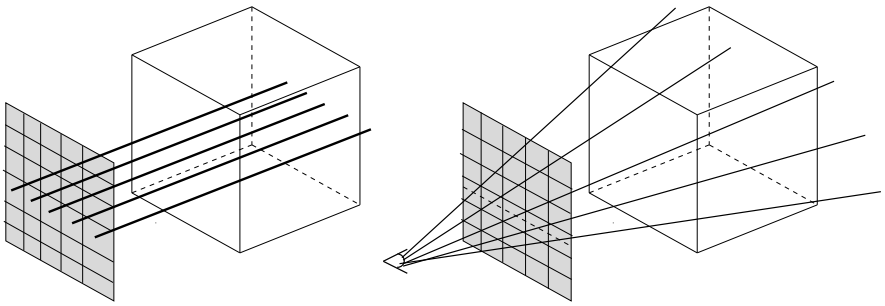
The intensity of the whole ray is given as

$$I_{approx}(s) = \sum_{i=0}^{n} C(s_i)\alpha(s_i) \prod_{j=0}^{i-1} \big(1 - \alpha_{\text{Sample}}(s_j)\big)^{\Delta x} \ , \tag{2.32}$$

where

$$\alpha_{\text{Sample}}(s_i) = 1 - \big(1 - \alpha(s_i)\big)^{\Delta x} \tag{2.33}$$

This is equivalent to compositing the samples. It is possible to choose more complicated functions as source term. Levoy [51] creates the impression of surfaces within a volume by using the gradient as "surface" normal in a Phong-based shading model. Alternative light models are evaluated analogously to the particle model. Ray segments are usually combined using the same equations as above, *i.e.*, by compositing samples. Light models only differ in the way in which intensity and opacity for a single segment (or sample) are computed.



(a) (b)

Figure 2.13: Projection types commonly used for direct volume rendering. For orthographic projection all rays are perpendicular to the image plane (a). For perspective projection all rays originate in an eye-point are are traced through pixels in the image plane. (b)

Ray-casting

Ray casting generates an image on a per pixel basis. For each pixel of the resulting image, one or more (if oversampling is used) rays are casted into the volume while evaluating a light model along each ray. Two projection types are commonly used. In *orthographic projection*, rays are perpendicular to the image plane. For *perspective projection*, a viewpoint is given in addition to the image plane, and rays are then traced from this viewpoint through each pixel of the image plane. As indicated in the previous section, evaluating the light model along a ray is usually done by replacing the integrals in the equation describing the light model by the Riemann sum approximation. Samples are placed uniformly along a ray, see Figure 2.14(a), partitioning the ray into segments. Within each ray segment, constant optical properties are assumed and intensity and opacity are computed. Individual ray segments are combined via compositing, see previous section. An important consideration for ray casting using uniform sampling is the distance between two samples. The necessary distance depends on the frequency of the "signal" along the ray, which in turn depends on cell size and additionally on the transfer function. For low-frequency transfer functions, one sample per cell is sufficient. Sample spacing should be equivalent to the radius of a sphere inscribed in the cell. For rectilinear cells this radius is equivalent to the smallest cell dimension. In general, it is beneficial to specify the sampling distance in terms of the radius of the inscribed sphere, as it makes a specification of a sampling rate more independent from the considered grid. High-frequency transfer functions may require more than one sample per cell.

Under certain circumstances a ray should be subdivided in segments that span only a single cell instead of segments of constant length. For example, when constant interpolation assigns one value to all locations in a cell, a light model can be evaluated analytically within individual cells. In this case, an enumeration of cells along a ray is desired, see Figure 2.14(b). A cell enumeration can be computed by extending Bresenham's algorithm, see Foley *et al.* [21], to the 3-d case.

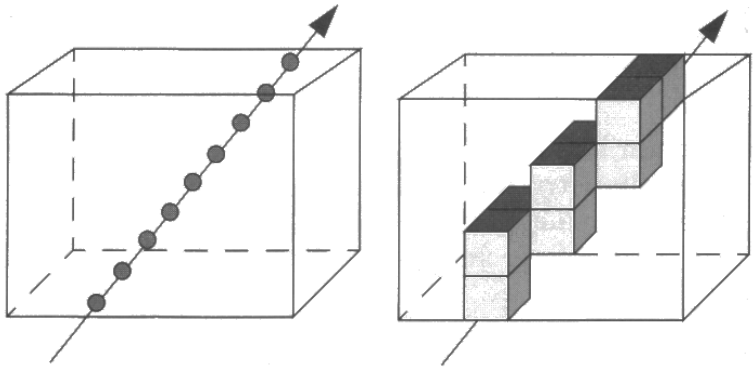


Figure 2.14: Ray casting evaluates the light model along a ray traced through a pixel. Commonly, samples are spaced uniformly along a ray (a). Variations exist that generate an enumeration of all cells along a ray (b). (Images courtesy of Schroeder *et al.* [74])

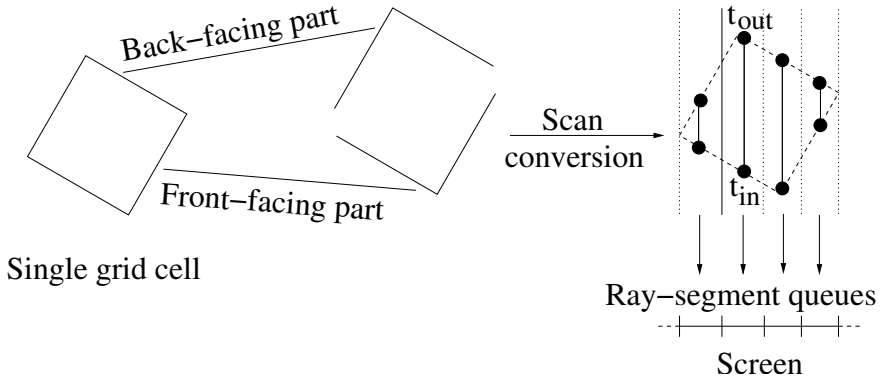


Figure 2.15: The cell-projection process.

Cell Projection

Cell projection [58] is an object-space volume rendering method. Similar to ray casting, rays are traced through the volume accumulating light along the way. Instead of tracing rays starting at pixels of the image plane, cell-projection methods construct these ray segments cell-by-cell. For each cell, all segments are constructed and merged with existing ray segments.

Ma's cell-projection approach [58] maintains a priority queue for each pixel that collects all ray segments contributing to that pixel. Figure 2.15 shows the fundamental idea of the cell-projection approach. Boundary faces of all cells are divided into three groups, *front-facing* faces (with normals directed toward the viewer), *back-facing* faces (with normals directed away from the viewer), and *view-perpendicular* faces (with normals perpendicular to the view direction). (View-perpendicular faces only exist when orthographic projection is used and are discarded as they do not contribute to the final image.) First, the front-facing faces are scan converted into a

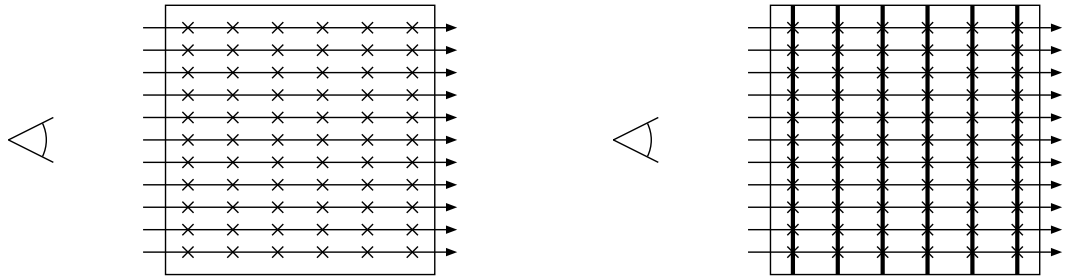


Figure 2.16: Accelerating ray casting using texture hardware. When uniform sampling is used, samples are located at equidistant positions along a ray (a). Considering the n-th sample on each ray, all samples form a plane (b). (Images courtesy of Oliver Kreylos)

buffer. For each pixel influenced by the cell, this buffer holds a depth corresponding to an entry parameter value t_{in} along the ray and an interpolated scalar value at that position. Subsequently, the back-facing faces are scan-converted. For each generated pixel, the depth corresponding to the exit parameter value t_{out} along the ray and an interpolated scalar value are computed. The entry parameter value t_{in} and corresponding scalar value for that pixel are read from the buffer, and the ray segment reaching from t_{in} to t_{out} is constructed. This ray segment is then inserted into the ray-segment queue of the corresponding pixel. If the ray segment is adjacent to existing ray segments in that ray-segment queue, it is merged with them. Two ray segments are adjacent when the union of their parameter intervals along the ray $\big([t_{0,\text{in}}, t_{0,\text{out}}]$ and $[t_{1,\text{in}}, t_{1,\text{out}}]\big)$ is continuous, *i.e.*, when either $t_{0,\text{in}} = t_{1,\text{out}}$ or $t_{0,\text{out}} = t_{1,\text{in}}$ holds. When all cells are processed, each ray segment queue contains only one ray segment which corresponds to the ray originating from the pixel. Ma considers tetrahedral cells using linear interpolation. As shown in Section 2.2.2, interpolated values in a linear tetrahedron can be obtained by linearly interpolating between values on the triangular boundary faces. If arbitrary cells are used, this property no longer holds. Additional information enabling interpolation must be stored instead of interpolated values. This will be discussed in detail in Section 5.2.4.

Williams *et al.* [90] introduced a similar scheme that constructs ray segments within cells. Instead of sorting ray segments using priority queues for all pixels, cells are pre-sorted, using, for example, the scheme of Max [61]. Then, cells are rendered in either back-to-front or front-to-back order. Newly generated ray segments are always adjacent to the previously computed ray segments and can be composited directly in the frame buffer. Using a piecewise linear transfer function, Williams *et al.* [90] evaluate the light model analytically for tetrahedra.

Hardware-accelerated Volume Rendering

Ray casting a scalar field over a rectilinear grid using uniform sampling can be accelerated utilizing standard graphics hardware [9, 77]. Figure 2.16(a) shows rays and samples generated when orthographic projection is used. In ray casting, samples are taken on a per-ray basis. When

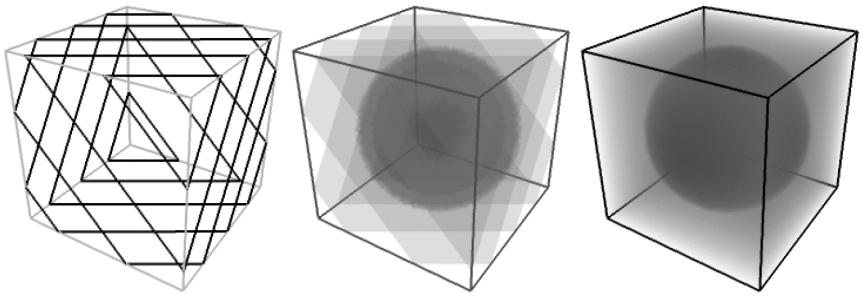


Figure 2.17: Simulating ray casting by rendering 3-d slices. The volume is sampled by rendering view-perpendicular slices clipped with the domain borders (left). Using a 3-d texture, color and opacity at each location on a slice are chosen according to a scalar value at that location (middle). By increasing the number of planes (*i.e.*, samples) a ray-casted image is computed. (Image courtesy of Westermann and Ertl [20])

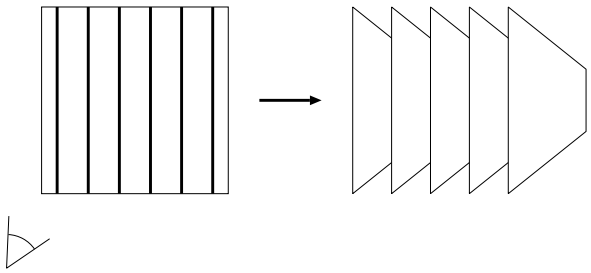


Figure 2.18: When axis-aligned slices are used instead of view-perpendicular slices, a viewing direction deviating from one of the three coordinate axis directions leads to visible artifacts in a rendered image. (Image courtesy of Oliver Kreylos)

considering the n-th sample along each ray, all those samples lie on a plane, see Figure 2.16(b). Since ray segments are combined using compositing, it is possible to generate an equivalent image to the one obtained using ray casting by rendering all these planes and compositing the result, see Figure 2.17. Colors and opacities are obtained from samples at the grid vertices and loaded as a 3-d texture. Subsequently, view-perpendicular planes, which are clipped with the domain borders, are rendered back-to-front. Interpolation and shading these planes is performed by the graphics hardware. Resulting images are composited using hardware alpha blending. When using the particle light model, it is possible to load scalar values directly into the 3-d texture. The transfer function is loaded into a 1-d texture. Transfer function lookup is achieved by cascading two texture lookup operations.

A simplified method uses three sets of axis-perpendicular slices — one for each coordinate axis. Based on the viewing vector, *i.e.*, the vector perpendicular to the image plane, the axis which is "most parallel" to the viewing direction, *i.e.*, the axis whose corresponding component

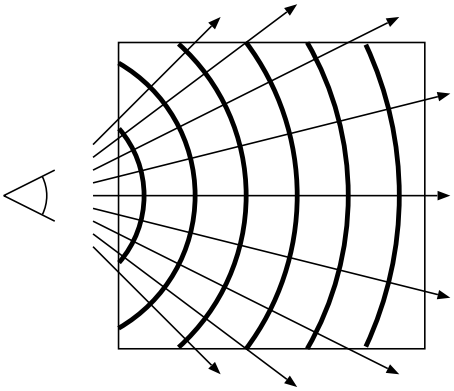


Figure 2.19: When using perspective projection, samples on rays lie on concentric spherical shells instead of planes.

of the viewing vector has the largest absolute value, is chosen. Subsequently, the corresponding set of axis perpendicular slices is rendered back-to-front using alpha-blending. The advantage of this method is that it can be used when only 2-d textures are available on a platform. However, using this method leads to artifacts in a rendered image when the viewing direction deviates from one of the three coordinate axis directions, see Figure 2.18. (When using perspective projection, sample positions lie on concentric spherical shells instead of planes, see Figure 2.19. It is possible to simulate perspective projection by rendering these speherical shells instead of view-perpendicular planes.)

2.2.5 Isosurface Extraction

Introduction

An isosurface is a surface representing all locations in 3-d space, where a trivariate scalar field $f(x, y, z)$ assumes a given isovalue v, *i.e.*, where $f = v$ holds. It partitions a 3-d volume into two distinct regions: locations "inside" an isosurface have an associated value greater-than or equal-to the isovalue; locations "outside" an isosurface have an associated value less than the isovalue. By varying the isovalue v, it is possible to visualize the entire scalar field. Isosurfaces are *implicit surfaces*, *i.e.*, surfaces given in the form $F(x, y, z) = 0$, where $F(x, y, z) = f(x, y, z) - v$. Desired is a surface that connects all locations, *i.e.*, where $F(x, y, z) = 0$ holds. Extraction of implicit surfaces was first used in computer graphics to display surfaces defined by mathematical functions. Later, implicit functions (in form of isosurfaces) were introduced in scientific visualization, which resulted in algorithms specialized for visualization purposes. The use of isosurfaces in scientific visualization started with medical data sets. CT and MRI scanners create data sets that are viewed as a set of slices along an axis, see Figure 2.20. Each slice is an image of constant resolution. The resolution along the "scan" axis, which is perpendicular to the slice images, is usually lower than within the slice imges. Initially, isosurface extraction schemes, see, for example, Christiansen and Sederberg [13], extracted contours in 2-d slices and then connected

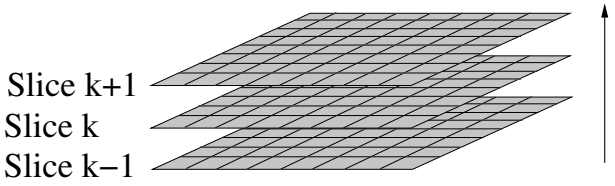


Figure 2.20: Data sets resulting from CT and MRI scanners are commonly viewed as a set of axis-aligned slice images.

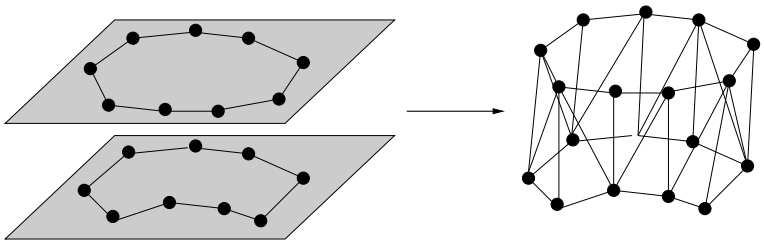


Figure 2.21: Initially, isosurface extraction schemes computed contours in slices and connected contours on adjacent slices using triangles.

these contours to form a triangulation of the isosurface, as illustrated in Figure 2.21.

Marching Cubes

The marching cubes (MC) method was introduced by Lorensen and Cline [56] in 1987 and has become the de-facto standard isosurface extraction algorithm in scientific visualization. MC assumes that data values are given at the vertices of a regular rectilinear grid. Lorensen and Cline explicitly mention medical data, given in form of slices, and connect samples from adjacent slices to form cubes. Today, medical volume data is already given in form of rectilinear cells (cubes). MC extracts an isosurface cell-by-cell in a divide-and-conquer approach. All cells are "marched" and in each "cube" (cell) a triangulation approximating the intersection of the isosurface with that cell is computed. This construction is performed locally and only depends on the values at the vertices of the current cell. In the remainder of this section, I will examine MC, emphasizing topological properties and aspects of the method. I do this as our methods for topology-based analysis of scalar fields, presented in Chapter 6, are motivated by this behavior and are only applicable if an isosurface extraction scheme preserves the topology of trilinear interpolation.

MC constructs a triangulation using the intersection points between the isosurface and the edges of a cell as vertices. Each cell vertex is classified as having either "positive" (*i.e.*, having an associated value larger-than or equal-to the isovalue v) or "negative" (*i.e.*, having an associated value less-than the isovalue v) polarity. An edge between two cell vertices has an intersection point when they differ in polarity. The edges that are intersected by the isosurface only depend on the vertex configuration, *i.e.*, which vertices are positive and which are negative. The original

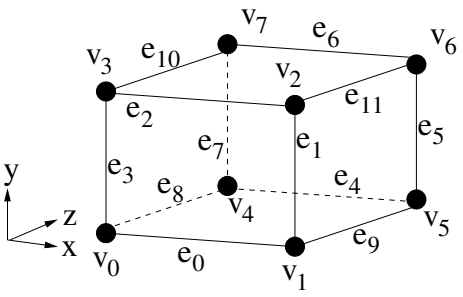


Figure 2.22: Numbering of vertices and edges of a hexahedral cell.

MC approach assumes that a unique triangulation for a given vertex configuration and stores triangulations in a look-up table (LUT). An LUT index is determined based on the polarities of a cell's vertices by numbering vertices from zero to seven, see Figure 2.22, and computing an LUT index using

$$\text{index} = \sum_{i=0}^{7} 2^i \begin{cases} 0 & \text{if vertex } i \text{ is negative, } \textit{i.e.}, v_i < v, \\ 1 & \text{if vertex } i \text{ is positive, } \textit{i.e.}, v_i \geq v. \end{cases}$$

This is equivalent to constructing an eight-bit integer with each bit corresponding to a vertex. If a vertex is positive, its corresponding bit is set. This view leads to an efficient implementation using bit operations. The resulting index references one of 256 possible triangulations in an LUT. Using rotational symmetry and "inversion" of vertex polarities (*i.e.*, exchanging the vertex classification for positive and negative) it is possible to reduce the 256 cases to 15 basic configurations shown in Figure 2.23.

Each triangulation consists of a list of indices of edges (edges in a cell are numbered from zero to eleven, see Figure 2.22) referencing the intersection point of that edge with the isosurface as vertex in the triangulation. Using a second LUT that contains a bit mask where each edge corresponds to a bit that is set if that edge is intersected, it is possible to efficiently precompute these intersection points.

Topological problems in MC were discovered by, for example, Dürst [17], who pointed out that MC can produce holes in isosurface triangulations. For certain configurations, contour topology in a cell is not completely determined by vertex polarities. Holes arise when a cell face implies ambiguous topology. Faces with alternating vertex polarities cause ambiguity problems. Three cases are possible, see Figure 2.24. Contours can separate negative vertices, see Figure 2.24(b), positive vertices, see Figure 2.24(c), or all vertices resulting in a non-manifold contour, see Figure 2.24(d).

All basis configurations of Lorensen and Cline's paper separate positive vertices of ambiguous faces. Identical triangulations are used for an inverse case. The resulting triangulation separates negative vertices. If a cell whose triangulation is derived from a basis configuration without inversion and a cell whose triangulation is derived from a basis configuration using inversion share an ambiguous face, a crack in the extracted isosurface triangulation arises, see Figure 2.25.

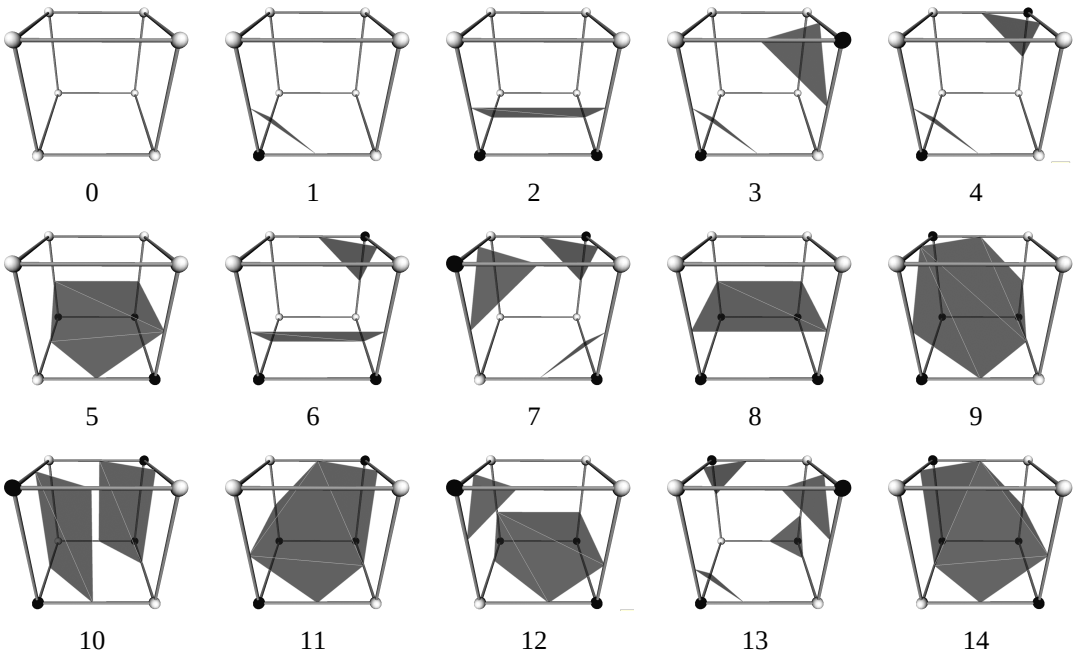


Figure 2.23: All 256 entries for the LUT used by MC can be constructed from the shown 15 base cases by using rotational symmetry and inversion.

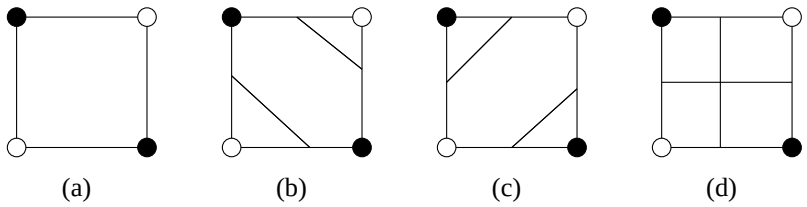


Figure 2.24: An ambiguous face configuration consists of vertex of alternating polarities (a). Different contour topologies on that face are possible, separating negative vertices (b), positive vertices (c), or all vertices (d) resulting in a non-manifold contour.

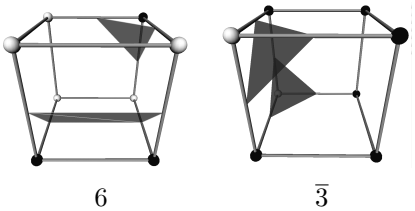


Figure 2.25: Adjacent cell configuration resulting in a crack in an isosurface extracted by an unmodified MC approach.

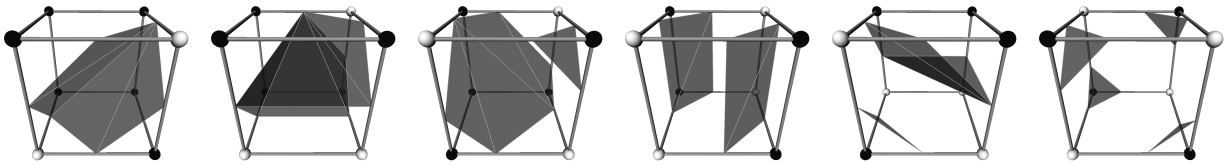


Figure 2.26: Additional cases used by implicit disambiguation.

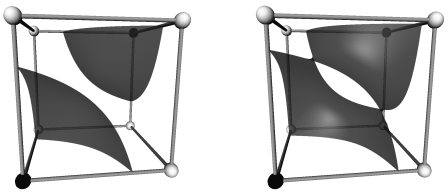


Figure 2.27: Trilinear interpolation within a cell. Opposite cell vertices can be separated (a) by two isosurface sheets or (b) connected by one isosurface.

Montani *et al.* [65] proposed an extended set of basis configurations, see Figure 2.26. For each MC basis configuration containing an ambiguous face, *i.e.*, basis configurations 3, 6, 7, 10, 12, and 13, they add its inverse to the set of basis configurations. The LUT created by their method, which is used, for example, in the Visualization Toolkit (VTK) [74], consistently separates positive vertices, preventing holes arising due to inconsistent topology on faces between cells. Closer examination shows that basis configurations $\overline{10}$, $\overline{12}$, $\overline{13}$ are redundant since it is possible to derive them from configurations 10, 12, 13 by means of rotational symmetry instead of inversion. Thus, extending the set of basis configurations from Figure 2.23 with configurations $\overline{3}$, $\overline{6}$ and $\overline{7}$ from Figure 2.26, and avoiding inversion for all basis configurations that contain an ambiguous face, will lead to a consistent case table.

Montani *et al.'s implicit disambiguation* chooses an arbitrary contour topology on ambiguous faces. In addition to ambiguous contour topology on a cell's face, there exist configurations where the topology within a cell is ambiguous. The two positive vertices in MC basis configuration 4, see Figure 2.27, for example, can either be separated or connected by a tunnel. While Montani *et al.'s* approach guarantees that a resulting isosurfaces has no cracks due to topological problems, *i.e.*, it produces a *consistent triangulation*, it fails to preserve correct isosurface topology, *i.e.*, it does not always produce a *correct triangulation*. This property causes problems when time dependent isosurfaces are considered as incorrect isosurface topology in a time step can result in abrupt topology changes.

To reproduce isosurface topology correctly, this topology must be defined, thereby providing a way to choose between possible configurations in ambiguous cases. Thus, producing topologically correct isosurfaces requires assumptions about the scalar function in a cell's interior. Hamann [31] and Nielson and Hamann [68] pointed out that bilinear interpolation on a cell's

faces is a natural extension to linear interpolation along edges and use bilinear contour topology to resolve ambiguities on faces.

The equation for bilinear interpolation on a face normalized to the unit square $[0,1] \times [0,1]$ is

$$bi(x,y) = (1-x)(1-y)v_0 + x(1-y)v_1 + xyv_2 + (1-x)yv_3 \ . \tag{2.34}$$

The nodal values v_0, v_1, v_2, and v_3 are located at $(0,0), (1,0), (1,1)$ and $(0,1)$, respectively. Equation 2.34 can be rewritten as

$$bi(x,y) = axy + bx + cy + d \ , \tag{2.35}$$

with

$$\begin{aligned} a &= v_0 - v_1 + v_2 - v_3 \ , \\ b &= v_1 - v_0 \ , \\ c &= v_3 - v_0 \text{ , and} \\ d &= v_0 \ . \end{aligned}$$

If a equals zero, the bilinear interpolant degenerates to linear interpolation and the contour is a straight line segment. No critical points and ambiguities exist. Otherwise, according to Equation 2.2, one can find critical points as points where the gradient vanishes. The partial derivatives of Equation 2.35 are

$$\frac{\partial bi}{\partial x} = ay + b \text{ , and} \tag{2.36}$$

$$\frac{\partial bi}{\partial y} = ax + c \ . \tag{2.37}$$

Setting Equation 2.36 and 2.37 to zero yields a unique solution at

$$\mathbf{p}_{\text{crit}} = \left(-\frac{c}{a}, -\frac{b}{a} \right) \ . \tag{2.38}$$

The type of the critical point can be determined by considering the Hessian, see Equation 2.3. The Hessian for f is

$$\begin{pmatrix} \frac{\partial^2 bi}{\partial x \partial x} & \frac{\partial^2 bi}{\partial x \partial y} \\ \frac{\partial^2 bi}{\partial y \partial x} & \frac{\partial^2 bi}{\partial y \partial y} \end{pmatrix} = \begin{pmatrix} 0 & a \\ a & 0 \end{pmatrix} \ .$$

The eigenvalues of the Hessian of bi at $\mathbf{p}_{\text{crit}}$ are

$$\begin{vmatrix} -\lambda & a \\ a & -\lambda \end{vmatrix} = \lambda^2 - a^2 \quad \rightarrow \lambda_{1,2} = \pm a \ . \tag{2.39}$$

Two eigenvalues of opposite sign exist for the Hessian at $\mathbf{p}_{\text{crit}}$ indicating that $\mathbf{p}_{\text{crit}}$ is a saddle of bi. The value of the saddle point is

$$bi\left(-\frac{c}{a}, -\frac{b}{a} \right) = \frac{da - bc}{a} = \frac{v_0 v_2 - v_1 v_3}{v_0 - v_1 + v_2 - v_3} \ . \tag{2.40}$$

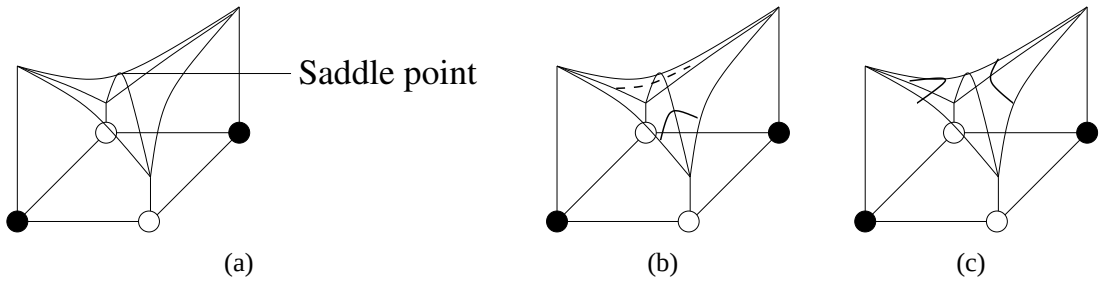


Figure 2.28: For an ambiguous face, the bilinear interpolant has a saddle (a). Contour topology depends on the relation of the isovalue with respect to the saddle value. For isovalues smaller than the saddle value, negative vertices are separated (b). For isovalues larger than the saddle value, positive vertices are separated (c).

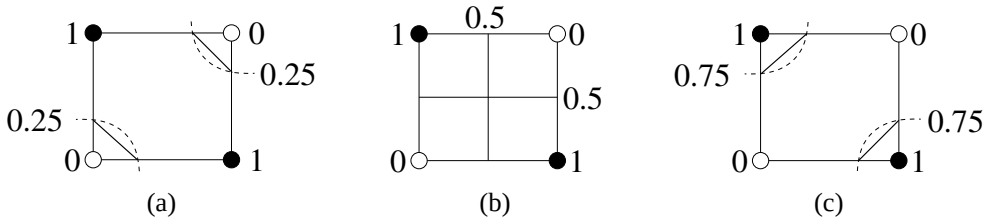
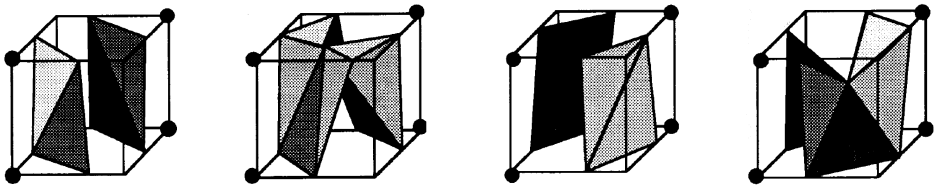


Figure 2.29: Ambiguous face: Considering bilinear interpolation contours for isovalues less than the saddle value are hyperbolic arcs separating negative vertices (a). For an isovalue equal to the saddle values, contours degenerate to a pair of axis aligned-asymptotes (b). For isovalues larger than the saddle value, contours are hyperbolic arcs separating positive vertices (c).

If bilinear interpolation does not reduce to linear interpolation, contours are a pair of hyperbolic arcs with coordinate-axis-parallel asymptotes that intersect in the saddle point. In the degenerate case, when the isovalue equals the value at the saddle point, these asymptotes become the contour. If the saddle point is inside a cell's face, isovalues exist for which both hyperbolic arcs intersect all four face edges, see Figure 2.28(a). In these cases, the contour on a face depends on the relation of the isovalue with respect to the saddle value. If the isovalue is smaller than the saddle value, the contour separates the negative vertices, see Figures 2.28(b) and 2.29(a). If the isovalue is larger than the saddle value, the contour separates the positive vertices, see Figures 2.28(c) and 2.29(c). If the isovalue is equal to the saddle value, the contour degenerates to the non-manifold set of axis-aligned asymptotes separating all vertices, see Figure 2.29(b). Nielson and Hamann determine connectivity for such ambiguous faces correctly by comparing isovalue and value at the intersection point of the asymptotes (which corresponds to the saddle point) and define an appropriately extended LUT. Ignoring the degenerate case, two choices for contour topology are possible on each ambiguous face of a configuration. Consequently, a basis configuration with f ambiguous faces has 2^f sub-configurations. MC basis configuration 10, for example, has two ambiguous faces, the bottom and the top face, resulting in four



(a) (b) (c) (d)

Figure 2.30: Sub-cases for MC basis configuration 10 as used by the asymptotic decider [31,68]. (Illustrations courtesy of Nielson and Hamann [68])

sub-configurations, see Figure 2.30. Nielson and Hamann also showed that for certain configurations, sub-configurations (b) and (d) of basis configuration 10, for example, it is necessary to use points in a cell's interior to generate a valid triangulation.

Nielson and Hamann [68] solved the problem of ambiguous faces, but did not consider ambiguities in a cell's interior, for example, the possible tunnel for MC basis configuration 4, shown in Figure 2.27. For this configuration, an isosurface can separate the opposite cell vertices or can "connect" them via a tunnel. To distinguish between the two cases, it is necessary to define isosurface topology in a cell's interior. Trilinear interpolation is commonly used in visualization, *e.g.*, for Direct Volume Rendering (DVR). Thus, the topology of contours of a trilinear interpolant lends itself as choice of "reference topology." A trilinear interpolant may contain up to two saddles in a cell's interior, see Nielson [67]. Similar to the determination of contour topology on a face, the topology of an isosurface within a cell can be determined by considering the relationship of a specific isovalue with regard to the values at saddle points. Several authors used the topology of trilinear interpolation to determine isosurface topology in a cell's interior, *i.e.*, the existence of tunnels.

The trilinear interpolant for the unit cube $[0,1] \times [0,1] \times [0,1]$ can be written as

$$T(x,y,z) = axyz + bxy + cyz + dxz + ex + fy + gz + h \; . \tag{2.41}$$

Considering the Hessian H_T of T

$$H_T(x,y,z) = \begin{pmatrix} 0 & az+b & ay+d \\ az+b & 0 & ax+c \\ ay+d & ax+c & 0 \end{pmatrix} \; , \tag{2.42}$$

its eigenvalues λ_i can be found as solutions to the equation

$$-\lambda^3 + k_x^2\lambda^2 + k_y\lambda^2 + k_z\lambda^2 + 2k_xk_yk_z = 0 \; , \tag{2.43}$$

where

$$\begin{aligned} k_x &= ax + c \; , \\ k_y &= ay + d \text{ , and} \\ k_z &= az + b \; . \end{aligned}$$

Since $H_T(x, y, z)$ is symmetric, three real-valued eigenvalues always exist. Examining Equation 2.43 leads to the following lemma.

Lemma 1 *A non-degenerate critical point of piecewise trilinear interpolation is always a saddle.*

Proof: By definition of a non-degenerate critical point, all three eigenvalues λ_1, λ_2 and λ_3 differ from zero. It is possible to write Equation 2.43 as

$$-(\lambda-\lambda_1)(\lambda-\lambda_2)(\lambda-\lambda_3) = -\lambda^3+(\lambda_1+\lambda_2+\lambda_3)\lambda^2-(\lambda_1\lambda_2+\lambda_1\lambda_3+\lambda_2\lambda_3)\lambda+\lambda_1\lambda_2\lambda_3 \ . \quad (2.44)$$

Comparing the coefficients of Equation 2.44 with those of Equation 2.43 yields the equation

$$\lambda_1\lambda_2 + \lambda_1\lambda_3 + \lambda_2\lambda_3 = 0 \ . \quad (2.45)$$

Since all λ_i differ from zero, it is not possible to satisfy this equation if all λ_i have the same sign. Consequently, one eigenvalue must have a different sign than the other two eigenvalues and the critical point is a saddle. □

The location of the saddle of trilinear interpolation can be calculated as

$$\mathbf{p}_{\text{saddle}} = \begin{cases} \left(-\frac{c}{a} \pm \frac{\sqrt{-a_x a_y a_z}}{a a_x}, -\frac{d}{a} \pm \frac{\sqrt{-a_x a_y a_z}}{a a_y}, -\frac{b}{a} \pm \frac{\sqrt{-a_x a_y a_z}}{a a_z}\right) & \text{if } a \neq 0; \\ \left(\frac{ce-bg-df}{2bd}, \frac{df-bg-ce}{2bc}, \frac{bg-ce-df}{2cd}\right) & \text{if } a = 0, \end{cases} \quad (2.46)$$

where $a_x = ae - bd$, $a_y = af - bc$, and $a_z = ag - cd$. Section 6.9.2 provides a derivation of Equation 2.46.

Natarajan [66] used saddle points (face saddles and an interior saddle) to determine the topology of an isosurface in a cell. His method only uses contour points on a cell's edges and no interior contour points, which produces incorrect triangulations in certain cases. Natarajan does not explicitly specify how to triangulate an isosurface according to his connectivity determination. He gives one example in form of Figure 5 in his paper. In that example, the triangulation in the lower-middle diagram is incorrect since a correct triangulation would require the use of internal points in a cell. An additional problem of this approach is that it assumes that there exists at most one saddle in a cell.

Chernyaev [11] used the asymptotic decider [31, 68] to resolve ambiguities on faces. Within a cell, saddle points are not considered explicitly. Instead, Chernyaev's method uses a criterion based on the asymptotic decider, see Figure 2.31. Chernyeav derived an analytical, quadratic function that traces the results of the asymptotic decider on bilinear slices within a cell. A tunnel exists if there is a slice that connects two areas of equal sign within the cell that are separated on the cell's faces. Chernyaev's approach uses additional points in a cell's interior to specify valid triangulations for some cases, *e.g.*, Case 7.3 (in his case numbering scheme [11]). However, it does not use additional points for all configurations that require them. This leads to several invalid triangulations. For example, case $6.1.2$ (in his numbering scheme) incorrectly separates positive *and* negative vertices on the cell's right side, see Figure 2.32.

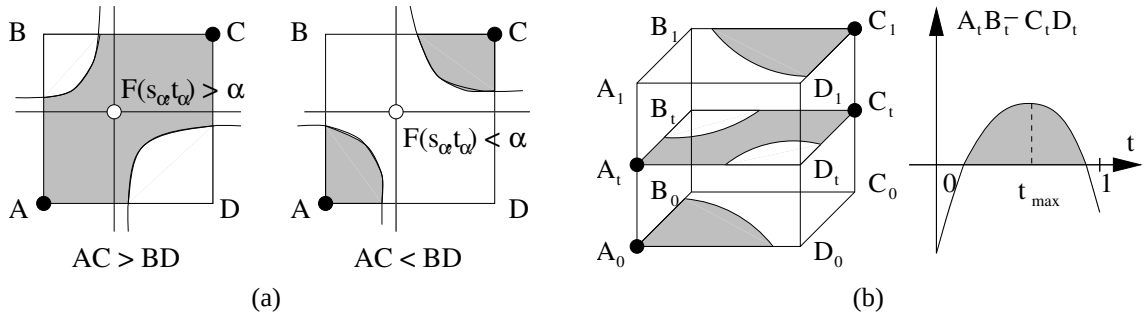


Figure 2.31: Ambiguity resolution according to Chernyaev [11]. On boundary faces the asymptotic decider [31, 68] used. Vertices are always ordered so that values A and C are larger than the isovalue and B and D are smaller than the isovalue (a). To resolve internal ambiguities, Chernyaev considers the results of the asymptotic decider on axis-perpendicular slices through the cell (b). (Illustrations courtesy of Chernyaev [11])

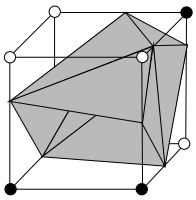
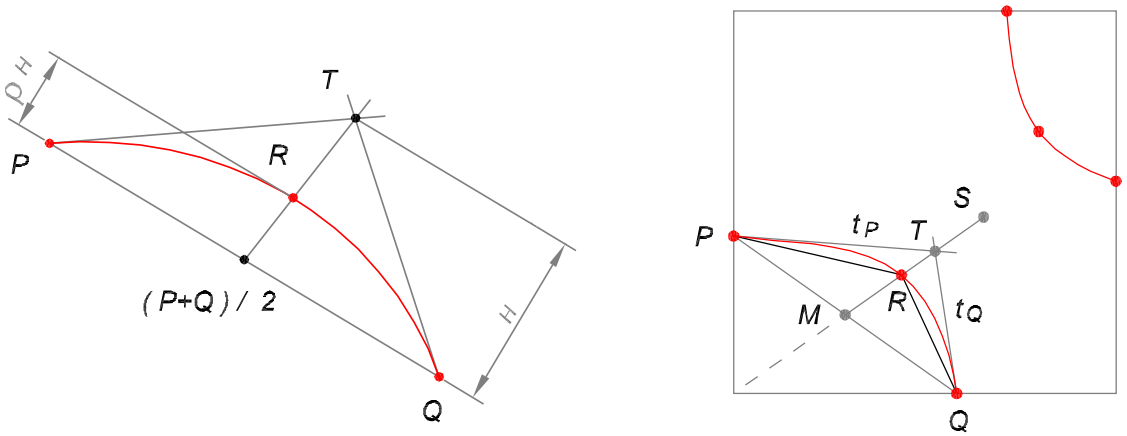


Figure 2.32: Even though Chernyaev's method accurately detects internal topology of a cell, his approach can produce invalid triangulations for some configurations, *e.g.*, basis configuration $6.1.2$ which separates positive and negative vertices on the right side of the cell.

Cignoni *et al.* [14] presented a hierarchical isosurface approximation approach. Their method recursively refines a triangulation obtained by a modified MC approach. To ensure that recursive refinement produces correct results, their method relies on topological correctness of the initial triangulation. Triangulations for ambiguous faces are described in terms of *open tunnels*. Additional points that are required to generate valid triangulations are obtained by intersecting these tunnels. Open tunnels are well suited to obtain initial triangulations for a refinement scheme. If they are used without subsequent refinement, the resulting triangulations are not ideal for isosurface representation. Still, the set of topological configurations underlying this method is incomplete. Differences in the number of considered cases in comparison to Chernyaev's work [11] have three reasons: First, Chernyaev's method uses inversion of vertex polarities to reduce the number of sub-cases for those cases that have four positive vertices. It always ensures that, on at least one boundary face, positive vertices are separated. Second, Cignoni *et al.* give two sub-cases for MC case number 6 (Table 6 in their paper) that are topologically equivalent. Last, when compared to the comprehensive work of Nielson [67], Cignoni *et al.* missed some sub-cases. Particularly, a sub-case of Configuration 13 in which a tunnel connects the positive vertices is missing (the complementary case to the case shown on the lower left side of Table 10



(a) (b)

Figure 2.33: Shoulder points used by Lopes [55] and Lopes and Brodlie [54]: A shoulder point on a conic curve connecting two points $\mathbf{P}$ and $\mathbf{Q}$ has a maximum distance from the line segment connection $\mathbf{P}$ and $\mathbf{Q}$. (Illustrations courtesy of Lopes [55])

in their paper). Furthermore, like Natarajan, Cignoni *et al.* consider only the possibility of one saddle point within the cell which can lead to incorrect results.

Topologically Correct Marching Cubes

In his dissertation, Lopes [55] discussed methods for improving "accuracy in scientific visualization" and introduced a MC variant that uses additional points on a cell's face and in its interior to improve the accuracy of an extracted isosurface. More recently, Lopes and Brodlie [54] described a slightly altered version of this approach using an altered and more concise terminology. I describe the original approach and its modification using the terminology from Lopes and Brodlie [54]. Lopes [55] improved the accuracy of isosurface extraction by using three types of additional points on a cell's faces and within its interior:

Face Shoulder Points The contour on a boundary face is a hyperbolic arc. MC approximates this hyperbolic arc with a line segment. It is possible to increase the accuracy of the approximation and better capture the "shape" of the hyperbola by using a polyline that includes the *shoulder point*. At the shoulder point the conic has maximum distance from the line segment, see Figure 2.33.

Tangent Points If a face is intersected twice by the same isosurface component, there exists a point within the cell, where the isosurface "loops" back to "leave" the cell by the same face by which it "entered" it, see Figure 2.34. At the corresponding *tangent point*, a slice parallel to that face is tangential to the isosurface.

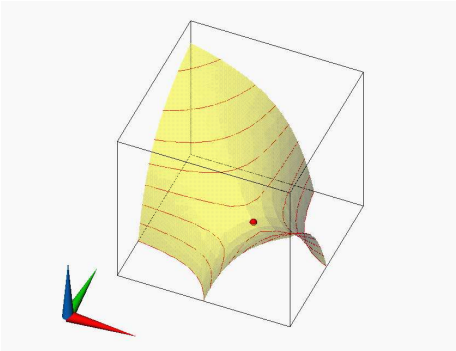


Figure 2.34: When a cell face is intersected twice by an isosurface (as the bottom face in the figure), a point within the cell exists, where the isosurface "loops back" to "leave" the cell by the same face by which it "entered" it. This occurs at a *tangent point* where the slice parallel to the cell face is tangential to the isosurface. On the tangential face, the contour degenerates to a pair of axis-aligned asymptotes.

Bi-shoulder points A bi-shoulder point provides an additional interior point for isosurface components that intersect each cell face at most once. A bi-shoulder point is located within a cell and is a shoulder point of the contours on two orthogonal, axis-perpendicular slices through the cell.

Lopes [55] constructed an isosurface in several subsequent steps: First, similar to the standard MC approach, a basis configuration is chosen based on vertex polarity. In addition to rotational symmetry and inversion, Lopes' approach reduces the number of base cases by considering "mirror" symmetry, *i.e.*, mirroring vertex configurations at axis-perpendicular faces dividing a cell at its middle, leading to 14 basis configurations (one less then the standard MC method), see Figure 2.36. By using the asymptotic decider on each ambiguous face, a sub-configuration is chosen that correctly reflects the topology on boundary faces, see Figures 2.37 – 2.42. To reduce the number of sub-cases, Lopes proved that certain sub-configurations of MC basis configurations cannot occur. If one considers two pairs of opposite faces, it is impossible that, for one pair, the positive vertices are separated and, for the other pair, the negative vertices are separated, see "crossed out" cases in Figure 2.42. For each configuration, the intersection of the isosurface with the cell is specified as a set of *boundary polygons*, which are read from an LUT. Resulting from connecting the edge intersection points along the boundary faces, a boundary polygon is specified as non-triangulated, not necessarily planar polyline that completely specifies topology on boundary faces of a cell. (In Lopes' original work, these boundary polygons are called *topological polygons*.) Boundary polygons are independent of interior cell topology, *i.e.*, the existence of tunnels. For example, the two quadrilaterals shown in Configuration 10 (a), see Figure 2.40, can either separate the diagonally opposite bold vertices or can connect them with a tunnel. In addition to a list of intersected edges, the entry for a boundary polygon in the LUT contains additional information specifying the number of vertices and the number of so-called "loop-back" faces intersected by the polygon. A loop-back face is a face that is intersected twice by the same

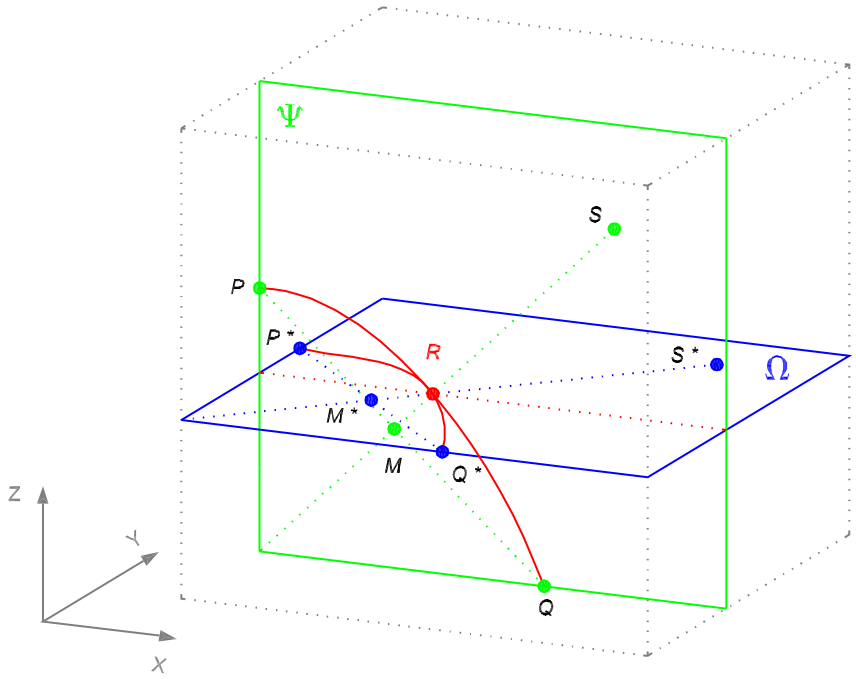


Figure 2.35: A bi-shoulder point within a cell's interior is a shoulder point on two perpendicular slices through the cell. (Illustration courtesy of Lopes [55])

boundary polygon, *i.e.*, that contains two edges of the same boundary polygon, see, for example, case 10 (b) in Figure 2.40. A trilinear isosurface corresponding to this case is shown in Figure 2.34. In this case, the isosurface loops back within the cell and leaves the cell by the same face by which it entered it.

After selecting the appropriate configuration, corresponding boundary polygons are refined by adding shoulder points of the conic (hyperbolic arc) implied by bilinear interpolants defined on cell faces, see Figure 2.33. The shoulder point can be computed as the intersection of a line connecting the midpoint of a linear contour approximation $\mathbf{M}$ and the location of the face saddle $\mathbf{S}$ with the contour, see Figure 2.33. Let $\mathbf{P}$ and $\mathbf{Q}$ be the intersection points of the hyperbolic arc with the boundary edges. The shoulder point of the hyperbolic arc lies on the line connecting the midpoint $\mathbf{M}$ of the line segment connecting $\mathbf{P}$ and $\mathbf{Q}$, *i.e.*, the linear approximation of the contour used by MC, and the intersection point of the two tangents to the hyperbolic arc in $\mathbf{P}$ and $\mathbf{Q}$. This line segment also runs through the saddle point of the bilinear interpolant. (If the bilinear interpolant does not have a saddle point, it degenerates to linear interpolation and no shoulder point exists as the hyperbolic arc degenerates to a straight line.) The parametric representation of the line $\overline{\mathbf{MS}}$ is

$$x = x_M + t\alpha \ , \qquad y = y_M + t\beta \ , \tag{2.47}$$

with

$$\alpha = x_S - x_M \ , \qquad \beta = y_S - y_M \ \text{, and} \qquad t \in [0, 1].$$

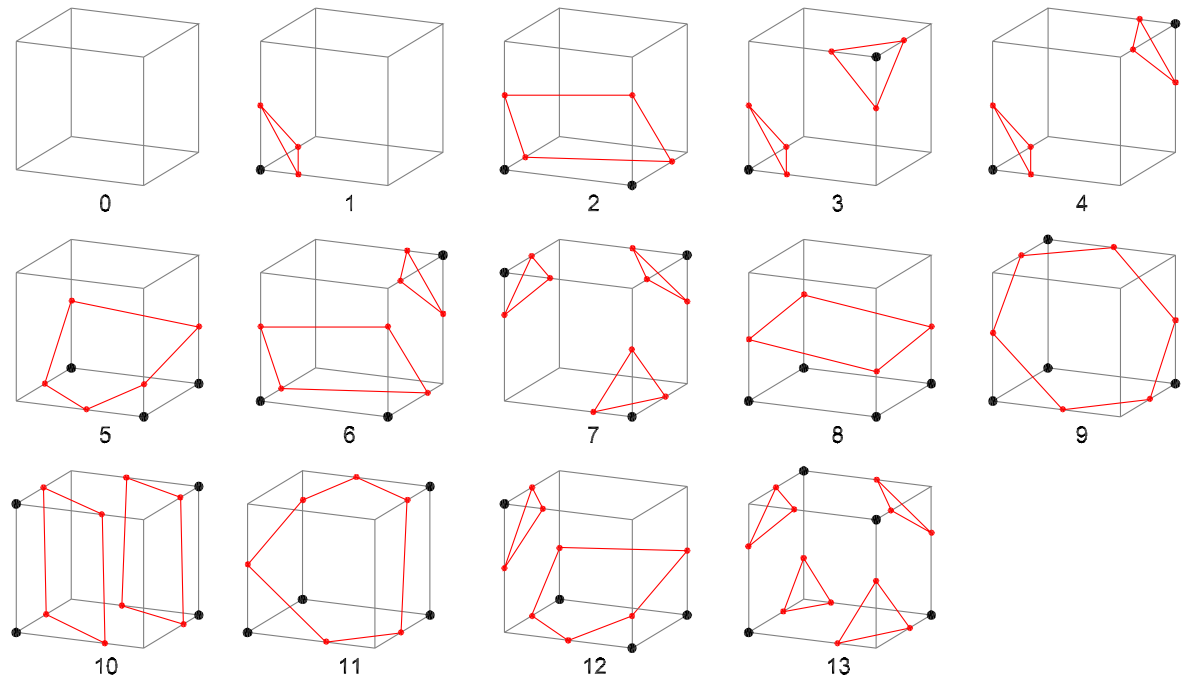


Figure 2.36: Boundary polygons for the MC base cases used by Lopes [55] and Lopes and Brodlie [54]. Note that MC basis configuration 14 is omitted as it can be obtained by mirroring MC basis configuration. (Illustration courtesy of Lopes [55])

Lopes shows [55] that t can be obtained as the solution of the quadratic equation

$$(c\alpha\beta)t^2 + (a\alpha + b\beta + c(x_M\beta + y_M\alpha))t + bi(x_M, y_M) = 0. \tag{2.48}$$

Solving Equation 2.48 results in two values for t. Using the value of $t \in [0, 1]$ in Equation 2.47, one can obtain the shoulder point $\mathbf{R}$. The shoulder point moves toward the saddle as the contour behavior approaches the degenerate case (two perpendicular lines). In the degenerate case, the shoulder point becomes the saddle point. By merging the shoulder points of the two hyperbolic arcs, an exact representation of a degenerate contour is possible. The gradual movement of two shoulder points toward the location of a face saddle supports a smooth transition between different topologies on a face.

In his dissertation Lopes [55] handled tunnels during the triangulation step. Each "enhanced" boundary polygon is triangulated individually by connecting its vertices to one or more internal contour points. Boundary polygons that contain at least one loop-back face cannot be part of tunnels and are always triangulated using tangent points. A tangent point is a point on the isosurface where the tangential plane to the isosurface is parallel to a loop-back face. The contour on the bilinear slice through the tangent point, which is parallel to the loop-back face, degenerates to a pair of axis-aligned asymptotes, see Figure 2.34. Thus, the tangential face can be determined as the face where the contour topology of a bilinear contour changes, *i.e.*, the face where the two

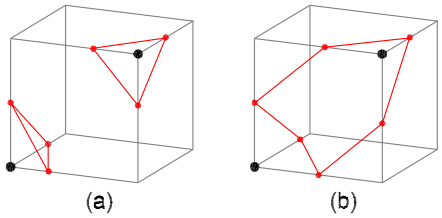


Figure 2.37: Boundary polygons for sub-configurations of MC basis configuration 3 used by Lopes [55] and Lopes and Brodlie [54]. (Illustration courtesy of Lopes [55])

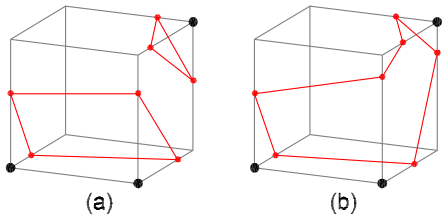


Figure 2.38: Boundary polygons for sub-configurations of MC basis configuration 6 used by Lopes [55] and Lopes and Brodlie [54]. (Illustration courtesy of Lopes [55])

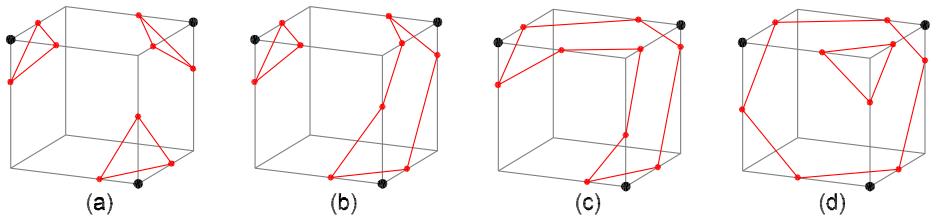


Figure 2.39: Boundary polygons for sub-configurations of MC basis configuration 7 used by Lopes [55] and Lopes and Brodlie [54]. (Illustration courtesy of Lopes [55])

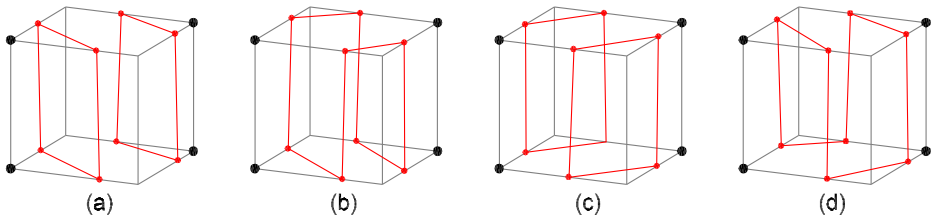


Figure 2.40: Boundary polygons for sub-configurations of MC basis configuration 10 used by Lopes [55] and Lopes and Brodlie [54]. (Illustration courtesy of Lopes [55])

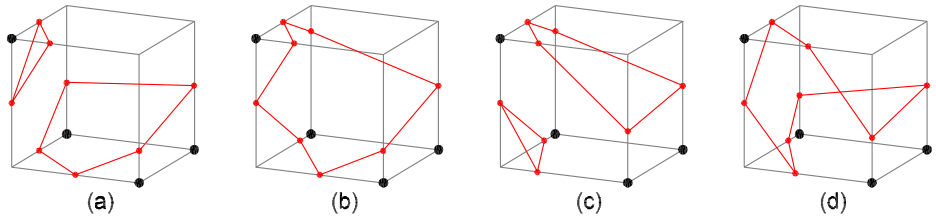


Figure 2.41: Boundary polygons for sub-configurations of MC basis configuration 12 used by Lopes [55] and Lopes and Brodlie [54]. (Illustration courtesy of Lopes [55])

partial derivatives of the trilinear interpolant, with respect to the two face parallel axes, vanish. All tangent points of a cell can be computed as points where at least two of the three partial derivatives $\frac{\partial F}{\partial x}$, $\frac{\partial F}{\partial y}$, $\frac{\partial F}{\partial z}$ vanish. Lopes [55] used this criterion to derive equations for the tangent points.

Figure 2.44 shows an example for triangulating a boundary polygon containing one loop-back face. All segments of the polyline forming the boundary polygon are connected to the tangent point belonging to that face via a triangle fan. Triangulations for polygons containing more than one loop-back face are more complicated. Lopes and Brodlie [54] provided an LUT for triangulations in Figure 12 of their paper. If a boundary polygon does not contain loop-back faces, two cases are possible:

1. All six tangent points lie in a cell's interior forming a polyline along the edges of a cuboid. A tunnel exists in the cell's interior, boundary polygons without loop-back faces are connected to the polyline formed by the inflection points.

2. A tunnel does not exist. A triangulation for a boundary polygon is obtained by connecting all its edges to a bi-shoulder point. A bi-shoulder point is a point that is a shoulder point on a pair of perpendicular planes passing through a cell and being parallel to coordinate-system planes. Lopes' method computes bi-shoulder points in an iterative approach by sweeping planes through a cell, starting from faces intersected by the boundary polygon. Since bi-shoulder points are not unique, a selection scheme is needed for two sweep faces that locates the most appropriate bi-shoulder point.

If a tunnel exists, all boundary polygons containing no loop-back face are triangulated by connecting them to the polyline defined by the tangent points. This approach yields correct results for all sub-configurations except sub-configuration $13b$. In his dissertation, Lopes stated that he did not observe any tunnels in any sub-configuration of case 13. However, in sub-configuration $13b$ two tunnels are possible: Either the positive or the negative regions can be connected by a tunnel. When a tunnel occurs, Lopes' original approach incorrectly connects all three polygons to form a tunnel, resulting in an invalid triangulation. Lopes and Brodlie [54] modifed the original approach to handle this case correctly. Tunnels are no longer detected based on the number of tangent points within a cell. Instead the isovalue is compared to the values of the saddles

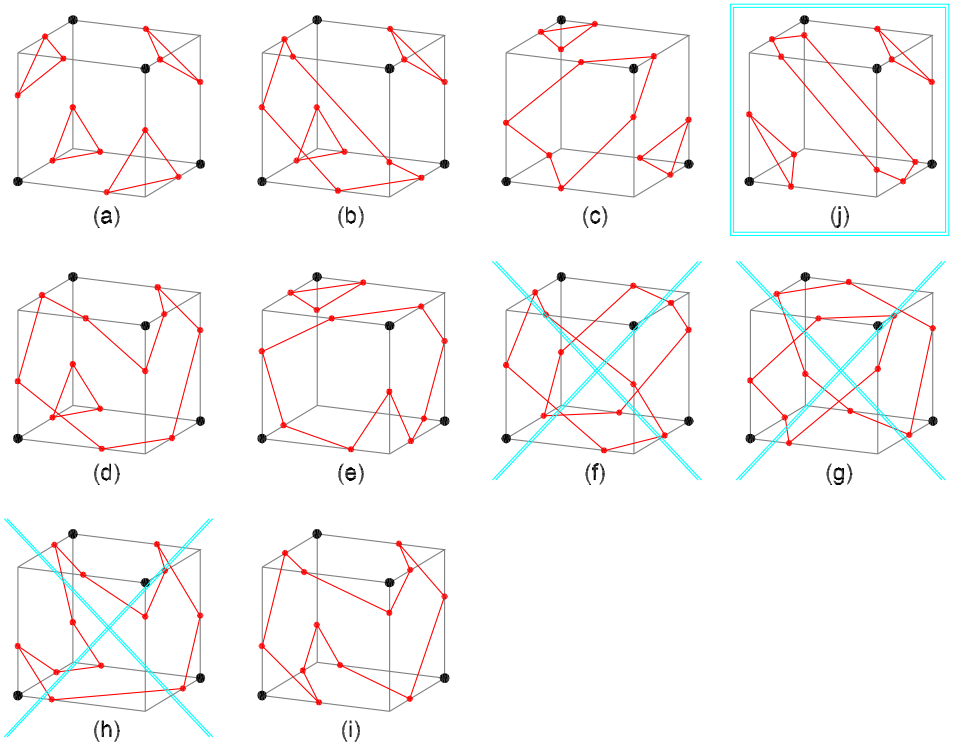


Figure 2.42: Boundary polygons for sub-configurations of MC basis configuration 13 used by Lopes [55] and Lopes and Brodlie [54]. Crossed out sub-configurations are not possible. The framed configuration is new with respect to Nielson and Hamann's asymptotic decider [31, 68]. (Illustration courtesy of Lopes [55])

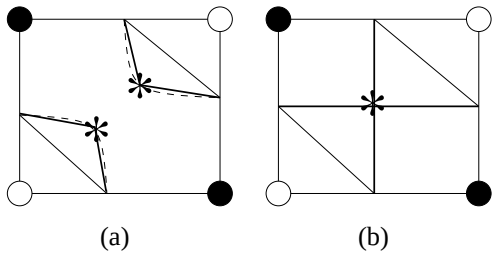
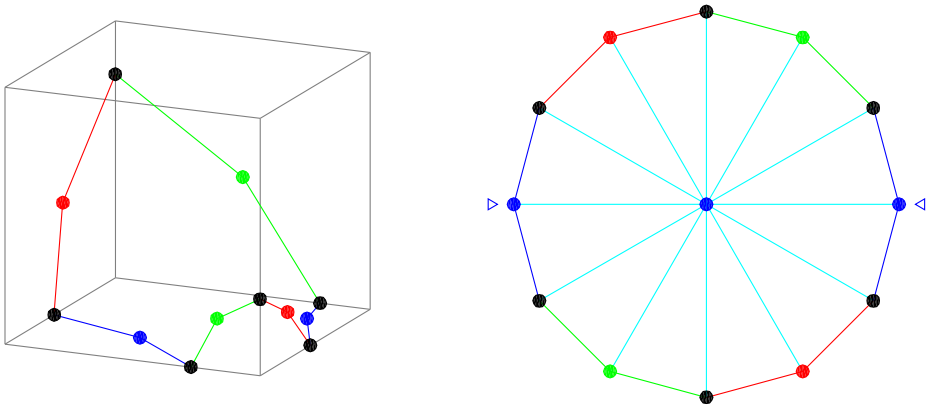


Figure 2.43: Incorporating shoulder points. (a) Using shoulder point, shown as "*," increases approximation quality of a contour (dashed) by replacing its one-segment approximation (solid) with a two-segment approximation (bold). (b) In the degenerate case, both shoulder points coincide with the location of the face saddle. If the shoulder points are merged, polyline approximations (bold) and contours coincide at the saddle. Without adding a shoulder point, the topology of a contour approximation is incorrect (solid).



(a) (b)

Figure 2.44: Triangulation of a boundary polygon consisting of six vertices containing one loop-back face. The bottom (z-perpendicular) face is the only loop-back face. In the resulting triangulation, all polyline segments on the cell boundary are connected to the corresponding tangent point T_z.

of the trilinear interpolant. Based on this decision, isosurface topology in a cell is completely determined before the triangulation begins.

Nielson [67] has recently provided a comprehensive analysis of the behavior of the trilinear interpolant. This analysis leads to an extension of the MC algorithm to accurately extract topologically correct contours of the trilinear interpolant. Differing from the original MC approach, Nielson does not use inversion to reduce the number of basis configurations. Unlike Lopes he also does not consider mirror symmetry to reduce the number of configurations. This leads to 22 basis configurations, see Figure 2.45. Nielson described two marching cubes variants: A consistent and a topologically correct variant. By specifying triangulations for all 22 basis configurations, Nielson's consistent approach creates a consistent case table similar to a case table obtained by using implicit disambiguation [65] approach. Nielson also described a topologically correct approach. Like Lopes [55] and Lopes and Brodlie [54], Nielson used additional points in the interior to obtain valid triangulations. Nielson' approach first determines the basis configuration based on vertex polarities. Subsequently, the asymptotic decider is used to determine topology on the cells boundary faces. To detect tunnels *DeVella's necklace* is considered. Nielson noted that axis-perpendicular planes exist where the intersection of the contour with that plane degenerates to two perpendicular, axis-aligned lines (asymptotes). DeVella's necklace comprises of the lines connecting the intersection points of these asymptotes. These intersection points are, in fact, the same points as the tangent points used by Lopes [55] and Lopes and Brodlie [54]. Similarly to Lopes' original approach [55], tunnels are detected by checking whether all these points are located inside the cell. Whenever internal points are necessary to obtain a valid isosurface triangulation, Nielson's approach uses points on DeVella's necklace. Unlike Lopes' method which uses all tangent points, Nielson's approach only uses the minimum

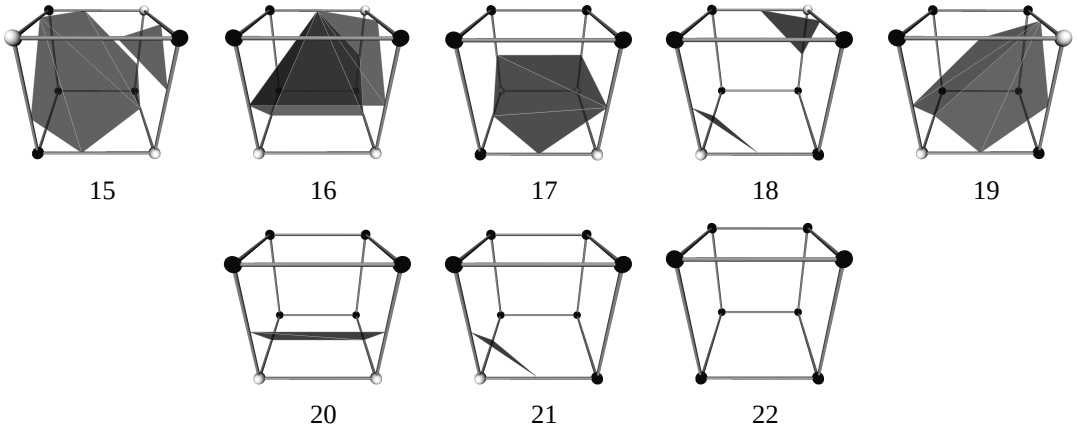


Figure 2.45: Additional basis configurations introduced by Nielson [67] by avoiding inversion in basis configuration reduction.

number of additional points to specify a valid triangulation. Nielson's method also distinguishes between the two possible tunnels in case 13 by comparing the isovalue to the value at the centroid of DeVella's necklace.

Lopes and Brodlie's [54] and Nielson's [67] approaches both produce correct isosurface topology for trilinear interpolation and specify correct triangulations for all resulting cases. While Nielson's method produces fewer triangles for all cases, Lopes' approach produces smoother transitions between different isosurface topology. Alternate topology definitions differing from the topology of the trilinear interpolant are possible. Van Gelder and Williams [27] defined the concepts of locally-linear and locally-quadratic configurations. A configuration is locally linear, if there is a linear function whose values at the cell's vertices result in the same classification of positive and negative vertices. It is locally quadratic if the same holds for a quadratic function. Van Gelder and Williams observed that MC basis configurations 1, 2, 5, 8 and 9 are locally linear and that all MC basis configurations, except basis configuration 13, are locally quadratic. Continuing from this observation, they considered values from quadratic functions to address topological correctness. Considering these functions, cells exist where the topology of a contour of the original function cannot be correctly determined based only on values at cell vertices. In fact, functions with different contour topology can result in cells having exactly the same vertex values. Van Gelder and Williams discussed several heuristics aimed at correctly reproducing the topology of quadratic functions, which, in their opinion, constitutes a satisfying local representation of the topology on ambiguous faces.

Alternate Isosurface Extraction Schemes

Hamann [32] described another, alternative approach to construct consistent triangulations in the interior of a cell by incorporating points into the triangulations that lie in a cell's interior and not exclusively on its edges. Hamann *et al.* [33] analyzed the exact behavior of contours on cell faces leading to a method that approximates a trilinear isosurface with rational-quadratic

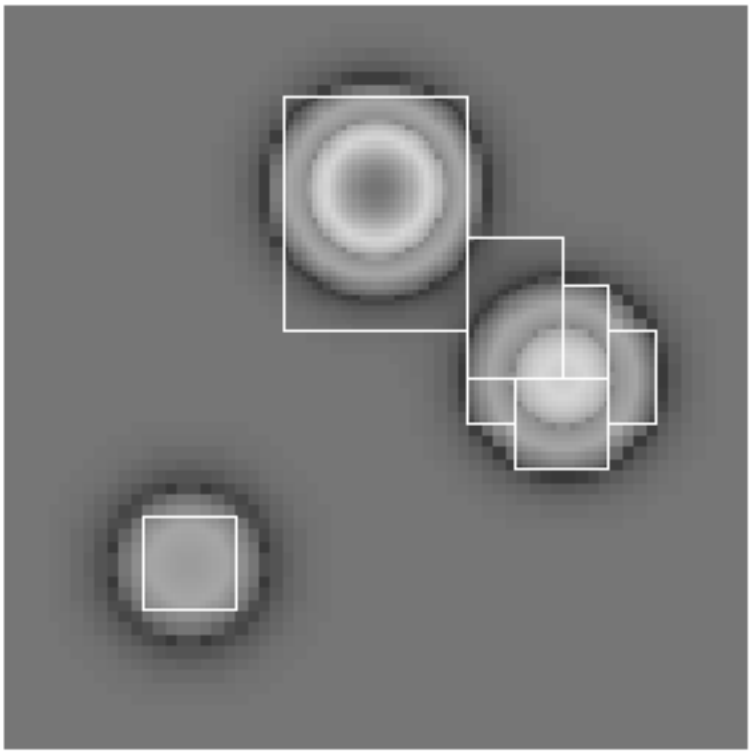


Figure 2.46: Visualization of AMR solution of a problem described by Poisson's equation.

Bézier patches. For their construction, they considered points lying on an isosurface and normals (gradients) at these points.

2.3 Adaptive Mesh Refinement

2.3.1 Introduction

Physical phenomena can vary widely in scale. Large regions in space can exist where a quantity varies only slightly, and thus can be adequately represented at a low resolution. Other regions may require higher resolutions to capture rapid changes. This characteristic is particularly noticeable in simulations of astrophysical phenomena that span several orders of magnitudes, covering vast regions of "empty" space. Physical phenomena are commonly simulated by considering a finite set of points in space connected by a mesh and discretizing the underlying equations accordingly.

Due to their simple topology, structured, rectilinear grids are frequently used in numerical simulations. While their implicitly given structure simplifies grid handling, allowing efficient parallelization, it also prevents adaption to localized changes in scale of the phenomenon. Consequently, a rectilinear grid which represents a physical phenomenon at adequate resolution in

high-variance areas constitutes a considerable waste of computational time in low-variance regions. Unstructured grids are highly adaptive to local resolution changes. However, they require the explicit storage of grid structure along with data values. Resulting algorithms are often complicated and difficult to parallelize.

Adaptive Mesh Refinement (AMR), introduced by Berger and Oliger [7] in 1984, combines the topological (structural) simplicity of regular, rectilinear grids with the adaptivity to local changes in resolution: The domain is discretized using one or more coarse grid grids in a *root level*. During the simulation, an error estimate is computed for each cell of this root level. All cells whose error estimate exceeds a given threshold are tagged for refinement. Subsequently, a set of rectilinear grids with an increased resolution is created, that overlaps all tagged cells. This is done recursively until all regions are represented at adequate resolution. Simulation results from higher-refined levels are propagated back to the coarser levels, resulting in a consistent representation of the domain. In Berger and Oliger's original approach [7], newly created grids are structured, rectilinear grids that can be rotated with respect to the parent level. Berger and Collela published a modified version [6] of this algorithm where newly created grids are axis aligned with respect to the parent level. AMR has also become increasingly popular outside the computational physics community. Today, it is used in a variety of applications. For example, Bryan [8] used a hybrid approach of AMR and particle simulations to simulate astrophysical phenomena. Figure 2.46 shows a 2-d AMR simulation of a problem described by Poisson's equation.

2.3.2 Berger-Colella AMR Data Format

Figure 2.47 shows a simple 2-d AMR hierarchy produced by the Berger-Colella method. The basic building block of a d-dimensional Berger-Colella AMR hierarchy is an axis-aligned, structured rectilinear grid. Each grid g consists of n_j hexahedral cells in each axial direction. I treat this number as an integer resolution vector $\mathbf{n}$[1]. The grid spacing, *i.e.*, the widths of grid cell in each dimension, is constant in a specific direction and given as a vector $\boldsymbol{\delta}_g = (\delta_{g,0}, \delta_{g,1}, \delta_{g,2})$, see Figure 2.47. Figure 2.47 illustrates that the distance between two samples is equal to the grid spacing. Each grid is positioned by specifying its origin $\mathbf{o}_g$. The simulation method typically applied to AMR grids is a finite-difference method. Consequently, a *cell-centered* data format is used, *i.e.*, dependent function values are associated with the centers of cells. Thus, the dependent function value associated with a cell $\mathbf{i}_g$, with $0 \leq i_{g,j} < n_j$, is located at

$$pos_{g,j}(\mathbf{i_g}) = o_{g,j} + \left(i_j + \frac{1}{2}\right)\delta_j \ , \tag{2.49}$$

see Figure 2.47. Since sample locations are implicitly given by the regular grid structure, it suffices to store dependent data values in a simple array using a fixed ordering scheme, *e.g.*, row-major order. The region covered by a grid g is denoted by Γ_g.

An AMR hierarchy consists of several levels Λ_l comprising one or multiple grids. All grids in the same level have the same resolution, *i.e.*, all grids in a level share the same cell width

[1] For convenience, the j-th component of a vector $\mathbf{x}$ is denoted as x_j.

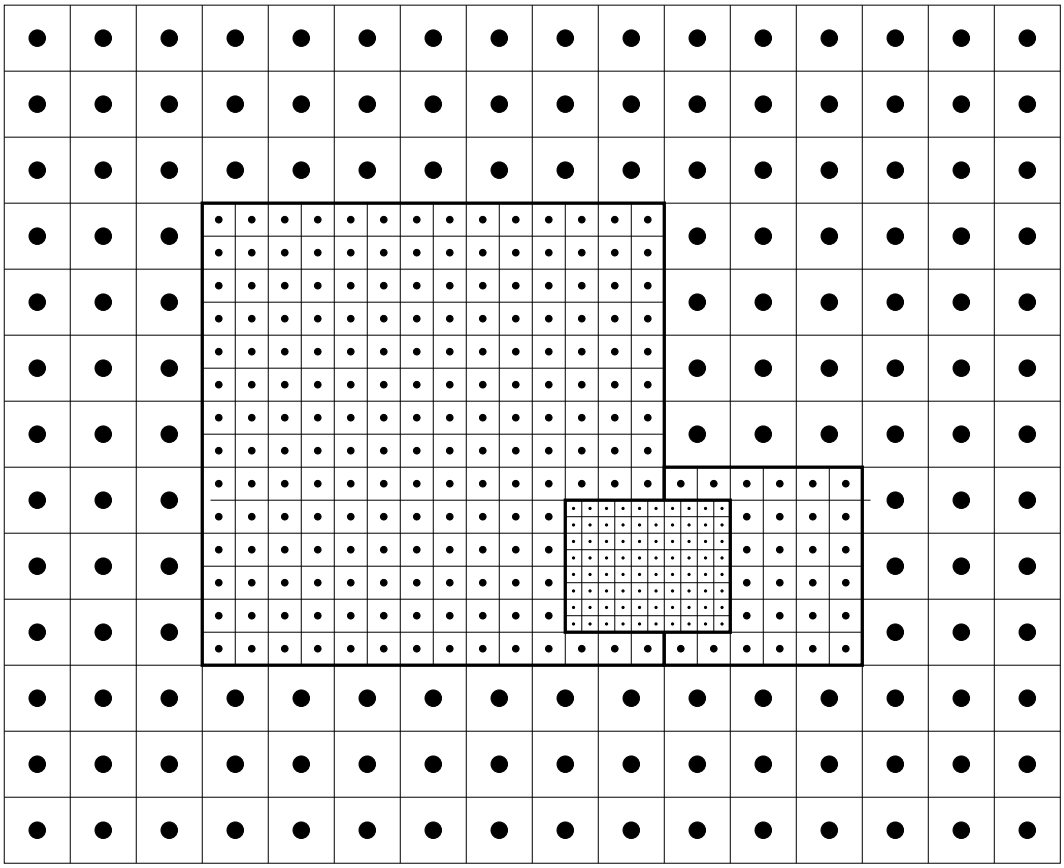


Figure 2.47: AMR hierarchy consisting of four grids in three levels. The root level consists of one grid. This grid is refined by a second level consisting of two grids. A fourth grid refines the second level, which overlaps both grids of the second level. Boundaries of the grids are drawn as bold lines. Locations at which dependent variables are given are indicated by solid discs.

vector $\boldsymbol{\delta}_g = \boldsymbol{\delta}_{\Gamma_l}$. The region covered by a level Γ_{Λ_l} is the union of regions covered by the grids of that level. In most AMR data sets, only the root level covers a contiguous region in space, while all other levels typically consist of several disjoint regions.

The hierarchy starts with the root level Λ_0, the coarsest level. Each level Λ_l may be refined by a finer level Λ_{l+1}. A grid of the refined level is commonly referred to as a *coarse grid* and a grid of the refining level as a *fine grid*. The *refinement ratio* r_l specifies how many cells of a fine grid contained in level l fit into a coarse-grid cell along each axial direction. (The refinement ratio is specified as a positive integer rather than a vector, as it is usually the same for all axial directions.) A refining grid can only refine complete grid cells of the parent level, *i.e.*, it must start and end at the boundaries of grid cells of the parent level. A refining grid refines an entire level Λ_l, *i.e.*, it is completely contained in Γ_{Λ_l} but not necessarily in the region covered by a single grid of that level. (This is illustrated in Figure 2.47, where the grid comprising the second level overlaps both grids of the first level.) Thus, in many cases, it is convenient to access grid

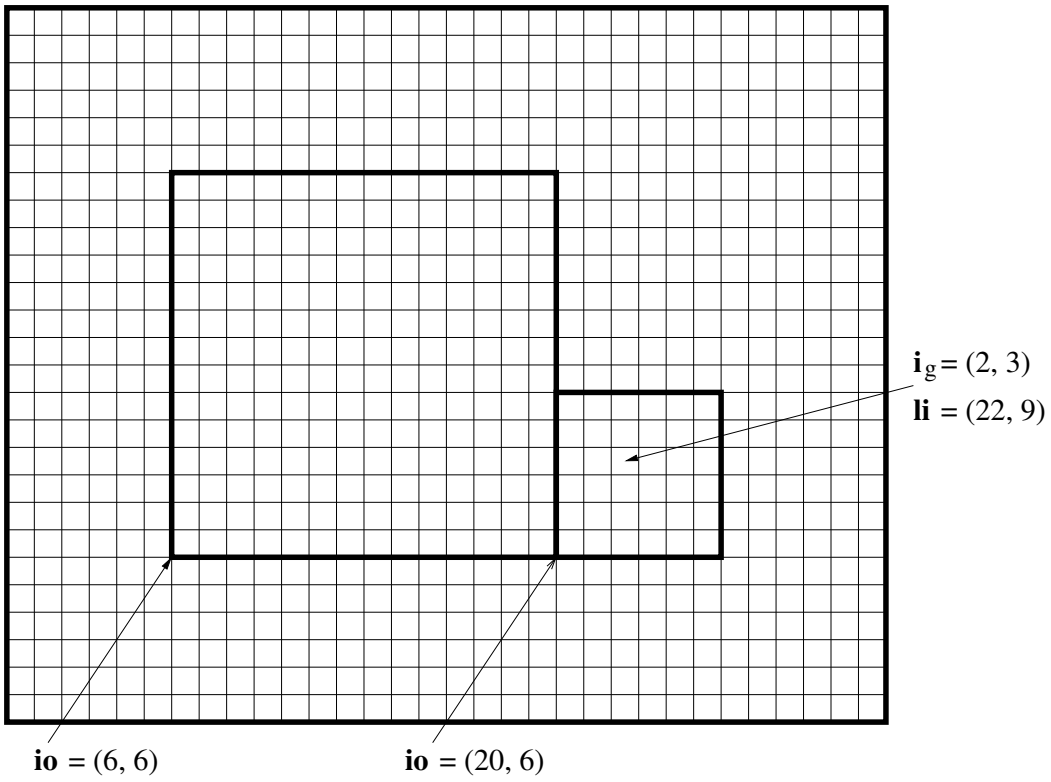


Figure 2.48: To specify the index of a cell with respect to a level rather than a single grid, it is assumed that an entire level is covered by a level grid with a cell width equal to $\boldsymbol{\delta}_{\Gamma_l}$. Individual grids within a level cover rectangular sub-regions of that level grid. An index of a grid cell can either be specified with respect to the level grid (**li**) or with respect to a grid containing that cell ($\mathbf{i}_g$). The integer origin of a grid is the level index of the grid cell with index **0** within the grid.

cells on a per-level basis instead of a per-grid basis.

To obtain the index of a cell within a level, it is assumed that a *level grid* with a cell width equal to $\boldsymbol{\delta}_{\Lambda_l}$ covers the entire domain Γ_{Λ_0}, see Figure 2.48. This level grid starts at the minimum extent of a bounding box surrounding the root level $\mathbf{o}_{\Lambda_0}$, where

$$o_{\Lambda_0,j} = \min\left\{o_{g,j} | \mathbf{o}_g \in \Lambda_0\right\} \ , \tag{2.50}$$

as shown in Figure 2.49. Since all grids in a level are placed with respect to boundaries of grid cells in a parent level, grid cells in the level grid coincide with grid cells in a grid of that level or are outside the region covered by the level Γ_{Λ_l}. The *level index* **li** of a grid cell is its index in the level grid. Using the level index, the origin of a grid in a level can be defined as an *integer origin*. The integer origin $\mathbf{io}_g$ of a grid g is the level index of the cell with index **0** within the grid, see Figure 2.48. Its components are defined by

$$io_{g,j} = \frac{(o_{g,j} - o_{\Lambda_0,j})}{\delta_{\Gamma_l}} \text{ , where } g \in \Lambda_l \ . \tag{2.51}$$

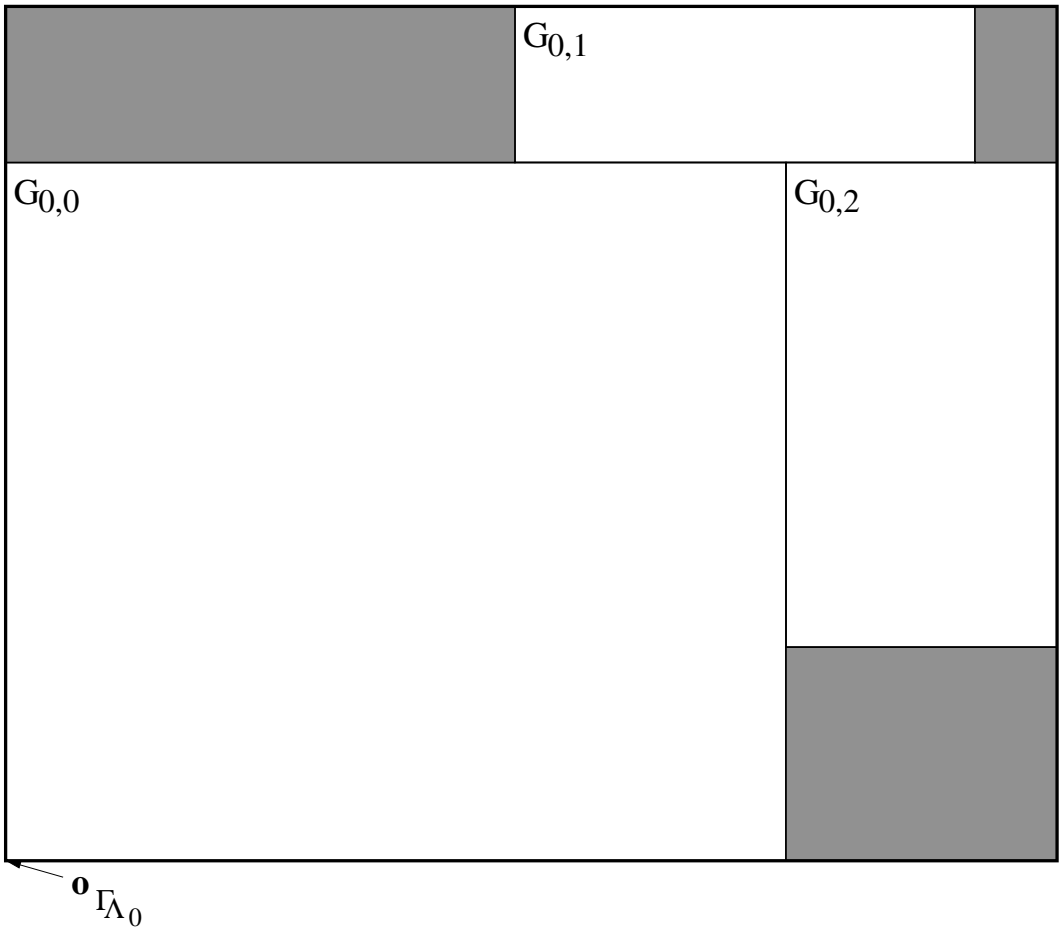


Figure 2.49: The AMR hierarchy shown in this figure contains three grids in the root level. Regions that are not covered by any grid are shaded. The origin of a level grid is the minimum extent of the bounding box enclosing all grids in the root level.

Since all grid cells of a fine grid must start and end at boundaries of cells of a coarse level, the components of the integer origin $io_{g,j}$ and the number of cells $n_{g,j}$ along an arbitrary axial direction of a grid g, belonging to level l, are always integer multiples of r_l. Individual grids in a level correspond to rectangular sub-regions of the level grid. To access a cell with a given level index $\mathbf{li}$, first the grid g that contains that level index has to be found, provided that such a grid exists. Second, the index $\mathbf{i}_g$ of the cell within that grid is obtained by subtracting this grid's integer origin $\mathbf{io}_g$ from $\mathbf{li}$.

The Berger-Colella scheme [6] requires a layer with a width of at least one grid cell between a refining grid and the boundary of the refined level. Bryan's scheme discards this requirement in his simulations.

Chapter 3

State of the Art

3.1 State of the Art of AMR Visualization

Initial work on AMR visualization focused on converting AMR data to suitable conventional representations and then visualizing these. Norman *et al.* [69] described a method for visualizing AMR data using standard toolkits like VTK [74], AVS[1] and IDL[2]. Their method converts an AMR hierarchy into an unstructured grid composed of hexahedral cells. The resulting unstructured grid is then visualized utilizing standard AVS, IDL, and VTK algorithms. By converting AMR data to an unstructured mesh, its main advantage, the implicit definition of grid connectivity, is lost. Overhead resulting from the required separate storage of grid structure results in poor scaling to large AMR data sets. Furthermore, this approach prohibits the use of the hierarchical nature of AMR data for efficient visualization algorithms. Recognizing these fundamental problems, Norman *et al.* continue by extending VTK to handle AMR as first-class data structure. They do not describe their extensions in detail, but cite slicing planes as an implemented example.

Max [61] described sorting schemes for cells during volume rendering including one method specifically geared toward AMR data. Ma [57] described and compared two approaches for parallel rendering of structured AMR data using the PARAMESH framework, see MacNeice *et al.* [60]. A PARAMESH hierarchy consists of grids of increasing resolution using a cell-centered data definition. PARAMESH grids are organized as blocks in a quadtree (in 2-d space) or an octree (in 3-d space) structure. Ma described two approaches for volume rendering of AMR data. One method resamples a hierarchy on an uniform grid at the finest resolution. The resulting grid is evenly subdivided and each part rendered on an individual processor. Computed images are composited using binary-swap composition, see Ma *et al.* [59]. A second method preserves the AMR structure. Individual blocks, *i.e.*, leaves of the octree, are distributed among the processors in a round-robin fashion to achieve static load balancing. Since a block structure can lead to many small ray-segments, Ma buffers these segments into larger messages to decrease communication

[1] Product of Advanced Visual Systems, see `http://www.vas.com/products/AVS5/avs5htm`.

[2] Interactive Data Language (IDL). Product of Research Systems, Inc., see `http://www.rsinc.com/idl/index.asp`.

overhead. Individual blocks are rendered using ray-casting. Two sampling schemes are used: A simple approach using a fixed, constant sample distance and an adaptive approach that decreases sample distance in finer resolution blocks.

Kreylos *et al.* [47] described a framework for remote, interactive rendering of AMR data. The framework consists of a "lightweight" viewer and a renderer running on one or several remote machines. The method of Kreylos *et al.* "homogenizes" an AMR hierarchy, *i.e.*, partitions it in blocks of constant resolution, using a k-d tree. The resulting blocks of constant resolution are distributed among processors and rendered using either a texture-based hardware-accelerated approach or a software-based cell-projection renderer. Two distribution strategies are implemented: One strategy attempts to distribute cost evenly among processors, the other variant tries to minimize data duplication.

Kähler and Hege [37, 38] introduced a method that partitions Berger-Colella AMR data into homogeneous regions and visualizes it using hardware-accelerated volume rendering based on textures. Their partitioning scheme aims to minimize the number of generated constant-resolution blocks utilizing a heuristic based on assumptions concerning the placement of refining grids by an AMR simulation. Generally, this approach generates fewer blocks than the approach described in this dissertation (which was published prior to Kähler and Hege's work [80]) and the approach described by Kreylos *et al.* [47], which is beneficial when data is rendered on a single machine. Kähler *et al.* [39] developed a method that uses AMR hierarchies for rendering sparse volumetric data. Given a transfer function, this method computes a transfer-function-specific AMR hierarchy for a volume data set. Their approach starts by representing the whole volume with a coarse mesh. Subsequently, all cells that contain data with an opacity exceeding a specified threshold are marked as relevant for refinement. The resulting AMR hierarchy is rendered using the algorithm of Kähler and Hege [37, 38]. The performance of the resulting algorithm is compared to an octree-based approach. Kähler *et al.* [36] used a set of existing tools to render results of a simulation of a forming star. Data was obtained by a simulation using the framework developed by Bryan [8]. Using *Virtual Director*, a virtual reality interface, camera paths were defined within a CAVE environment. Animations of the AMR simulation were rendered using the hardware-accelerated approach of Kähler and Hege [37, 38]. To enhance depth perception, rendered images are augmented with a background that is obtained by rendering a particle simulation of the formation of the early universe.

By specifying a transfer function, and a range of isovalues, Park *et al.* [70] produced volume-rendered images of AMR data based on hierarchical splatting, see Laur and Hanrahan [50]. Their method converts an AMR hierarchy to a k-d-tree structure consisting of blocks of constant resolution. Each node of this k-d tree is augmented with an octree. Octree and k-d-tree nodes contain a 32-bit field, where each bit represents a continuous range of isovalues. Using the k-d tree and the octree, regions containing values within the specified range are identified and rendered back-to-front using hierarchical splatting.

Ligocki *et al.* [53] described *ChomboVis* [3], a framework for the visualization of hierarchical computations using AMR. Their paper contains a brief summary of AMR visualization efforts to

[3] Joint effort of the Applied Numerical Algorithms Group and of the Visualization Group at LBNL. See `http://seesar.lbl.gov/anag/chombo/chombovis.html` for further details.

date.

3.2 State of the Art of Topology-based Data Analysis

Topology of level sets for data given on simplicial meshes using linear interpolation in cells has been a subject of research in computational geometry. Related to our work is the concept of *contour trees* that encode topological changes of level sets. A contour tree is a graph that describes how contours in a level set appear, join, split, or disappear when changing the isovalue. Van Kreveld *et al.* [78] used contour trees to speed up isosurface extraction. Using a contour tree, their approach computes a minimum seed set for isosurface reconstruction. Starting from the resulting seed points, all isosurface components are tracked and computed. Carr *et al.* [10] introduced a new scheme for efficient computation of a contour tree. Their method constructs a join and a split tree and combines them into a contour tree. Join trees are constructed starting from the largest value (highest peak) and adding points one at a time in order of decreasing height. While adding points, a set of growing contours is traced. Whenever two contours merge, the nodes in the join tree that correspond to their peaks are connected and subsequently treated as a unit. Split trees are computed by starting at the smallest value (lowest valley) and adding points in order of increasing height.

Bajaj *et al.* [1] also considered tetrahedral meshes. They introduced a *contour spectrum* specifying properties like 2-d contour length, 3-d contour area and gradient integral as functions of the isovalue. The contour spectrum is displayed along with the contour tree of a data set aiding a user in identifying interesting isovalues. Bajaj *et al.* [3] also developed a technique that uses topology to enhance visualizations of trivariate scalar fields. Their method employs a C^1-continuous interpolation scheme for rectilinear grids and approximations for the first and second derivative to detect critical points of a scalar field. Subsequently, integral curves (tangent curves) are traced starting from locations close to saddle points. These integral curves are superimposed onto volume-rendered images to convey structure of the scalar field.

Topology is also important in the context of data simplification to preserve important features of a data set. Bajaj and Schikore [2] extended previous methods to develop a compression scheme preserving topological features. Their approach detects critical points of a piecewise-linear bivariate scalar field. In this approach, "critical vertices" are those vertices for which the "normal space" of the surrounding triangle platelet contains the vector $(0, 0, 1)$. Integral curves are computed by tracing edges of triangles along a "ridge" or "channel." Bajaj and Schikore's method incorporates an error measure and can be used for topology-preserving mesh simplification.

Fujishiro *et al.* [24] used a *hyper-Reeb graph* for exploration of scalar fields. A Reeb graph encodes topology of a surface, see Section 2.1.2. A hyper-Reeb graph encodes changes of topology in an extracted isosurface. For each isovalue that corresponds to an isosurface topology change, a node in the hyper-Reeb graph exists that contains a Reeb graph encoding the topology of that isosurface. Fujishiro *et al.* [24] constructed a hyper-Reeb graph using "focusing with interval volumes," an iterative approach that finds a subset of all critical isovalues, which has been introduced by Fujishiro and Takeshima [25]. Considering just the images shown in their paper, it seems that their approach does not detect all critical isovalues of a scalar field. Initial

work [24, 25] used a hyper-Reeb graph for automatic generation of transfer functions. Fujishiro *et al.* [26] extended this work and used a hyper-Reeb graph for exploration of volume data. In addition to automatic transfer function design, their extended method allows them to generate translucent isosurfaces between critical isovalues.

Gerstner and Pajarola [28] defined a bisection scheme that enumerates all grid points of a rectilinear grid in a tetrahedral hierarchy. Using piecewise-linear interpolation in tetrahedra, critical points can be detected using a criterion derived from Banchoff's work [4], see Section 2.1.2. Data sets are simplified by specifying a traversal scheme that descends only as deep into the tetrahedral hierarchy as necessary to preserve topology within a certain error bound. The method incorporates heuristics that assign importance values to topological features, enabling controlled topology simplification.

Kraus and Ertl [46] used topology guided downsampling to simplify data sets while considering their topological behavior. Based on [28] they developed a heuristic indicating whether a vertex in a rectilinear regular grid is an ordinary point, minimum, maximum, or saddle. This heuristic is used to guide a down-sampling scheme while trying to preserve some topological properties of the original data set.

Konkle *et al.* [45] introduced a method to compute Betti numbers β_0 to β_2 for isosurface triangulations. Loosely defined, β_0 is the number of connected components, β_1 is the number of independent tunnels, and β_2 is the number of closed regions defined by a two-manifold surface in three-dimensional space. Konkle *et al.* show, that Betti numbers provide valuable information to a user that can be displayed along with an isosurface.

Pascucci and Cole-McLaughlin [71] extended work on contour trees. Based on the contour-tree construction algorithm presented in [10] they developed an algorithm that computes an augmented contour tree containing additional nodes corresponding to genus changes of contour components and specifies Betti numbers at each node. They subsequently developed a divide-and-conquer algorithm for contour-tree construction that improves efficiency. The central part of this new algorithm is independent from a specific interpolant used within grid cells. The authors used this ability to extend their algorithm to trilinear interpolation on hexahedral cells.

Cox *et al.* [15] considered topological changes of isosurfaces extracted by the *Spider Web* algorithm [41, 42]. Using linear interpolation, the Spider Web algorithm determines intersection points along edges of all cells intersecting the isosurface. Each face of a cell has either two or four intersection points of its edges with the isosurface. If a face has two intersection points, the Spider Web algorithm assumes that they are connected. If a face has four intersection points, a criterion similar to the asymptotic decider [68] is used to determine which edge intersection points are connected. In a second pass, a so called articulation point is computed for each connected set of intersection points within a cell. An isosurface triangulation is obtained by connecting the articulation point to all pairs of connected intersection points. By analyzing the results of the Spider Web algorithm, Cox *et al.* [15] developed a set of criteria to detect critical points in a volume data set when a so called admissible interpolant is used within cells. Both, isolated critical points and "critical isosets" (or regions) are detected. Cox *et al.* considered vertices to detect critical points or isosets. A single vertex is classified using a criterion similar to that used in this work. Critical isosets are detected by considering vertices that are edge connected to the isoset and counting connected components. Cox *et al.* also considered and detected saddles

on cell faces. However, their approach does not allow for critical points in a cell's interior which are ruled out by their admissibility criterion. Thus, their approach cannot be used for scalar fields defined by interpolation, and it ignores any possible topological changes within cells. It furthermore does not consider extended face saddles. After detecting critical points and critical isosets, Cox *et al.* devloped a method that partitions a volume into regions of equal topological behavior using a criticality tree which is related to a contour tree.

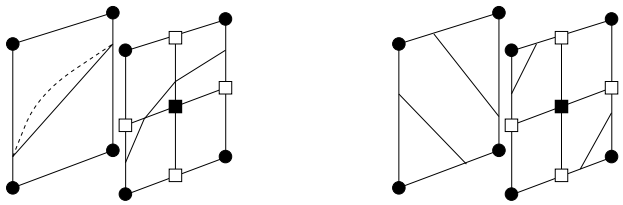
Chapter 4

Crack-free Isosurfaces for Adaptive Mesh Refinement Data

4.1 Introduction

Due to the hierarchical nature of AMR simulations, AMR data lend themselves to hierarchical visualization. One of the problems encountered, when isosurfaces are extracted using an MC method, is the cell-centered AMR format. MC methods expect that dependent data values are associated with a cell's vertices instead of its center. One possibility to deal with this incompatibility is using a resampling step to compute values at the vertices of the grid and convert the data to a vertex-centered format. If a resampling step is used to replace the values at a cell's center with values at its vertices, "dangling" nodes arise at level boundaries. Even if a consistent interpolation scheme is used, *i.e.*, one that assigns the same value to a dangling node as is assigned to its location in the coarse level, a crack in an isosurface extracted by MC can result, see Figure 4.1. MC approximates contours on boundary faces by line segments connecting intersection points of the contour with a face's edges. When a coarse face and a subdivided face with a dangling node are adjacent, two differing approximations of the bilinear contour are used on the same face. On the coarse face, the contour is approximated by a line segment connecting its intersection points with the face's edges. On refined face, a line segment is used to approximate the contour on each "sub-face," resulting in a polyline capturing the bilinear contour at a higher resolution. Coarse and fine contour only match up, when all vertex values are collinear and the bilinear contour degenerates to a line segment. In general cases, this does not happen, thus causing a crack between coarse and fine approximation of the bilinear contour, see Figure 4.1(a). An additional problem arises, when an MC approach based on implicit disambiguation is used. If the coarse face has an ambiguous configuration, the arbitrary choice of contour topology used by implicit disambiguation may be incorrect. Subdivision of the face in the finer level can resolve the ambiguity, and contour topology on the refined face is correct. Topology of coarse and fine contour differ, resulting in a crack, see Figure 4.1(b).

Using an MC approach that reproduces correct bilinear topology on boundary faces, for example, the asymptotic decider [68], avoids cracks due to inconsistent contour topology. Cracks



(a) (b)

Figure 4.1: Whenever dangling nodes occur, an isosurface extracted by MC can contain cracks, even if the dangling nodes has the same value as interpolation assigns to it in the coarse level. (a) MC approximates contours on boundary faces by line segments. The polyline on the subdivided face does not match up with the line segment on the coarse face, resulting in a crack. (b) Implicit disambiguation chooses arbitrary contour topology on ambiguous faces. When refining a face resolves the ambiguity, contour topology on the refined face may differ from contour topology on the coarse face.

due to differences in approximation accuracy are also encountered by approaches that try to decimate the number of triangles produced by marching cubes by using grids at different resolutions. Shekhar *et al.* [76] used an octree for hierarchical data representation. By adaptively traversing the octree and merging cells that satisfy certain criteria, their method reduces the amount of triangles generated for an isosurface. Their scheme removes cracks in a resulting isosurface by moving fine-level vertices at boundaries to a coarser level to match up contour approximations with those of the coarse level. Westermann *et al.* [88] modified this approach by adjusting the traversal criteria and improving the crack-removal strategy. Their method replaces triangles in the coarse cell by a triangle fan that matches up with the polyline in the finer level.

Since in the case of AMR data dangling nodes only occur because of the resampling to the vertex-centered case, it is best to avoid resampling completely. This is achieved by interpreting the locations of the samples, see Equation 2.49, as vertices of a new grid that is "dual" to the original one. Cell centers become the vertices of the vertex-centered *dual grid.* The implied connectivity information between these centers is given by the neighborhood configuration of the original cells.

The dual grids for the first two levels of the AMR hierarchy shown in Figure 2.47 are shown in Figure 4.2. It is important to note, that the dual grids have "shrunk" by one cell in each axial direction with respect to the original grid. The result is a gap between the coarse grid and the embedded fine grids. Due to the existence of this gap, there are no dangling nodes causing discontinuities in an isosurface. However, to avoid cracks in extracted isosurfaces as a result of gaps between grids, a tessellation scheme is needed that "stitches" grids of two different hierarchy levels.

I fill those gaps in an index-based stitching step. Vertices, edges and faces of a fine grid are connected to vertices in the coarse level using a look-up table (LUT) for the possible refinement configurations. The resulting stitch mesh consists of tetrahedra, pyramids, triangular prisms and hexahedral cells. By extending an MC method to these additional cell types, I define an

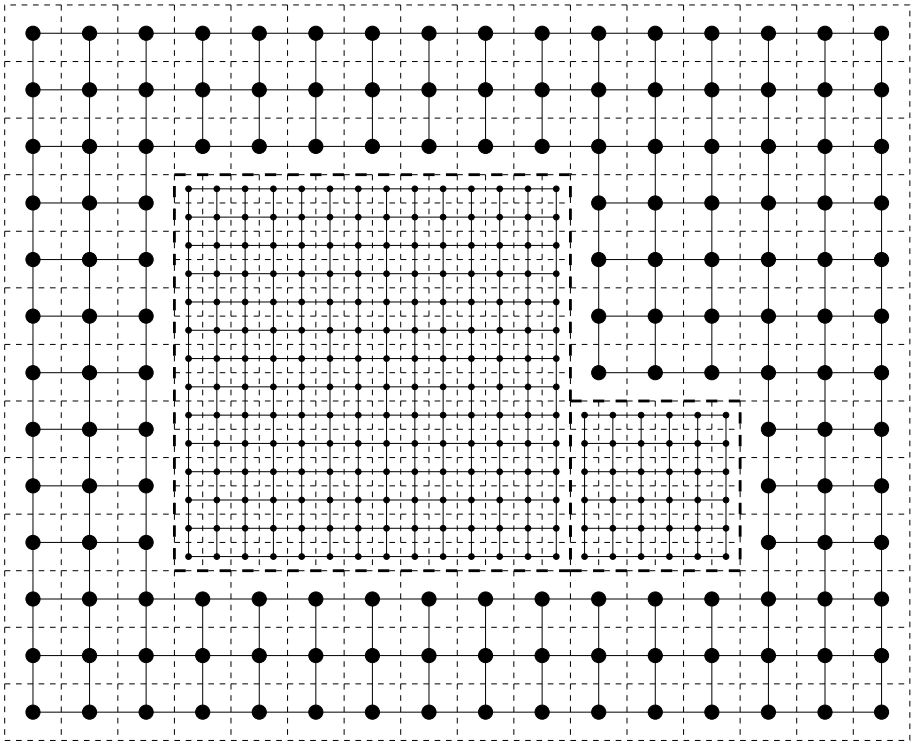


Figure 4.2: Dual grids for the three AMR grids comprising the first two hierarchy levels shown in Figure 2.47. The original AMR grids are drawn in dashed lines and the dual grids in solid lines

isosurface extraction scheme that avoids cracks in an isosurface at level boundaries.

The original Berger-Colella scheme [6] requires a layer with a width of at least one grid cell between a refining grid and the boundary of the refined level. Even though Bryan [8] eliminates this requirement, my method still requires it. This is necessary, as this requirement ensures that only transitions between a coarse level and the next finer level occur in an AMR hierarchy. Allowing transitions between arbitrary levels would force it to consider a large number of cases during the stitching process. (The number of cases would be limited only by the number of levels in an AMR hierarchy, since transitions between arbitrary levels are possible.) These requirements are equivalent to requirements described by Gross *et al.* [30] who also do not permit transitions between arbitrary levels.

4.2 Stitching 2-d Grids

A *stitch mesh*, used to fill gaps between different levels in the hierarchy, is constrained by the boundaries of the coarse and the fine grids. In order to merge levels seamlessly, the stitch mesh must not subdivide any boundary elements of the existing grids. In the 2-d case, this is achieved by requiring that only existing vertices are used and no new vertices generated. Since one of the

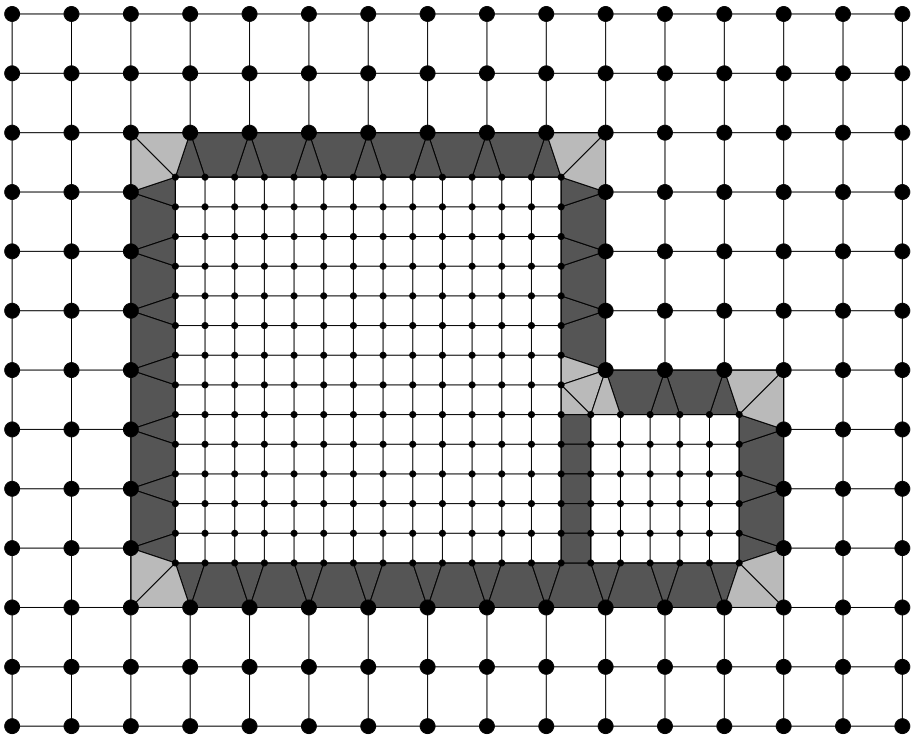


Figure 4.3: Stitch cells for first two levels of AMR data set shown in Figure 4.2

reasons for using the dual grids is to avoid the insertion of new vertices, whenever possible, this causes no problems.

In the 2-d case, a constrained Delaunay triangulation, see, for example, Chew [12], can be used to fill the gaps between grids. For two reasons, I do not do this. First, while in the 2-d case only edges must be shared between the stitching grid and the dual grids, entire faces must be shared in the 3-d case. The boundary faces of rectilinear grids are rectangles that cannot be shared by tetrahedra without being subdivided, thus causing cracks when used in an MC-based isosurface extraction scheme. Second, an index-based approach is more efficient, since it takes advantage of the regular structure of the boundaries, while avoiding problems that might be caused by this regular structure when using a Delaunay-based approach.

The stitching process for a refinement ratio of two is shown in Figure 4.3. Stitch cells are generated for edges along the boundary and for the vertices of the fine grid. The stitch cells generated for the edges are shown in dark gray, while the stitch cells generated for the vertices are drawn in light gray. For the transition between one fine and one coarse grid, each edge of the fine grid is connected alternatingly to either a vertex or an edge of the coarse grid. This yields triangles and quadrilaterals as additional cells. The quadrilaterals are not subdivided, since subdivision is not unique. (This in turn would cause problems in the 3-d case when quadrilaterals become boundary faces shared between cells.) The vertices are connected to the coarse grid via two triangles. A consistent partition of the deformed quadrilateral is possible. The obvious

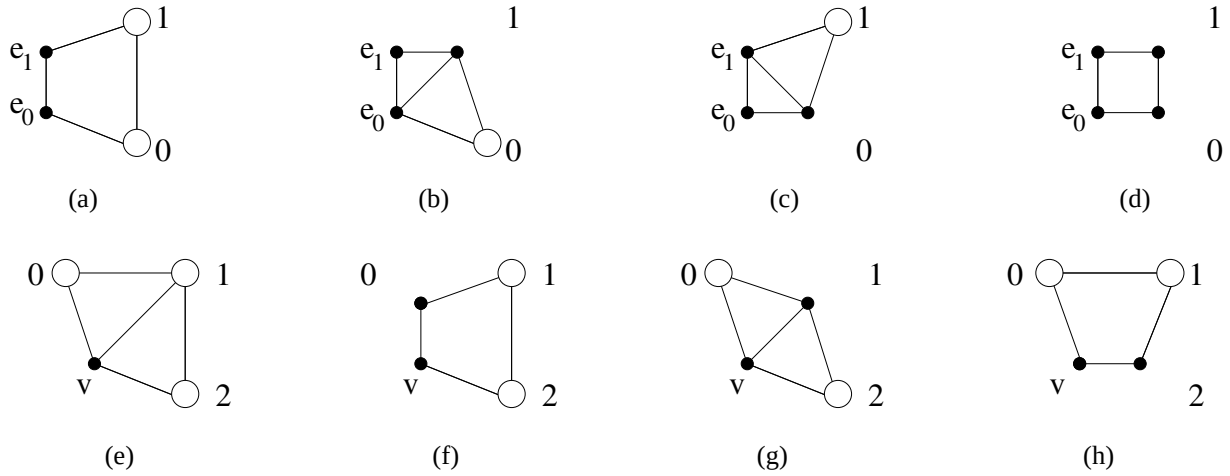


Figure 4.4: Possible cases for connecting a boundary edge $\overline{e_0 e_1}$, cases (a)-(d), or a boundary vertex v, cases (e)–(h), to a coarse grid. If cells of the coarse grid are refined, the coarse grid points (circles) are replaced by the corresponding refining point (solid black discs)

choice is to connect each vertex to the two coarse edges that are "visible" from it.

In the case of multiple grids, a check must be performed: Are the grid points in the coarse grid refined or not? If a fine edge is connected to a coarse point, this check is simple. If the coarse point is refined, the fine edge must be connected to another fine edge; this yields a rectilinear instead of a triangular cell. The case of connecting to a coarse edge is more complicated and is illustrated in Figures 4.4 (a)–(d). If both points are refined, see Figure 4.4(d), the fine edge is connected to another fine edge. As a result, adjacent fine grids yield the same cells as a "continuous" fine grid. Problem cases occur when only one of the points is refined, see Figures 4.4(b) and 4.4(c). Even though it is possible to skip these cases and handle them as vertex cases of the other grid, a more consistent approach is to include them in the possible edge cases. However, the same tessellations should be generated for both cases, as shown in Figure 4.4.

The cases arising from connecting a vertex are illustrated in Figures 4.4 (e)-(h). In addition to replacing refined coarse grid points by the nearest fine-grid point, adjoining grids must be merged. If either of the coarse grid points 0, see Figure 4.4(f), or 2, see Figure 4.4(h), is refined, it is possible to change the border vertex to a border-edge segment by connecting it to the other refined grid point and treating it as an edge, and using the connection configurations from the previous paragraph, *i.e.*, those shown in Figures 4.4 (a)–(d). (This case occurs along the bottom edge of the fine grids shown in Figure 4.3.)

Even though arbitrary integer-refinement ratios r_l are possible for AMR grids, refinement ratios of two and four are the most common ones used. The stitching process can be generalized to more general refinement ratios. Instead of connecting edge segments of the refining grid alternatingly to a coarse-grid edge segment and point, $(r_l - 1)$ consecutive edge segments must be connected to one common coarse-grid point. Every r_l-th fine edge must be connected to a coarse edge. The same connection strategy results from connecting each fine grid edge to a parallel

"phantom edge" that would exist if the grid continued in that direction. If both end points of the phantom edge are within the same grid cell in the parent level, the edge is connected to the coarse grid point within that cell. If the phantom edge crosses a boundary between two coarse grid cells, the fine edge is connected to the edge formed by the two grid points in those cells.

Even though the valence of the grid points of the coarse grid is increased, this is not a problem with the commonly used refinement ratios. Furthermore, it is important to note that arbitrary refinement ratios would not add more refinement configurations. The fundamental connection strategies remain the same. A fine-grid edge is connected to a coarse-grid vertex or a coarse-grid edge. A fine-grid vertex is connected to two coarse-grid edges. The cell subdivisions shown in Figure 4.4 can be used for arbitrary refinement ratios.

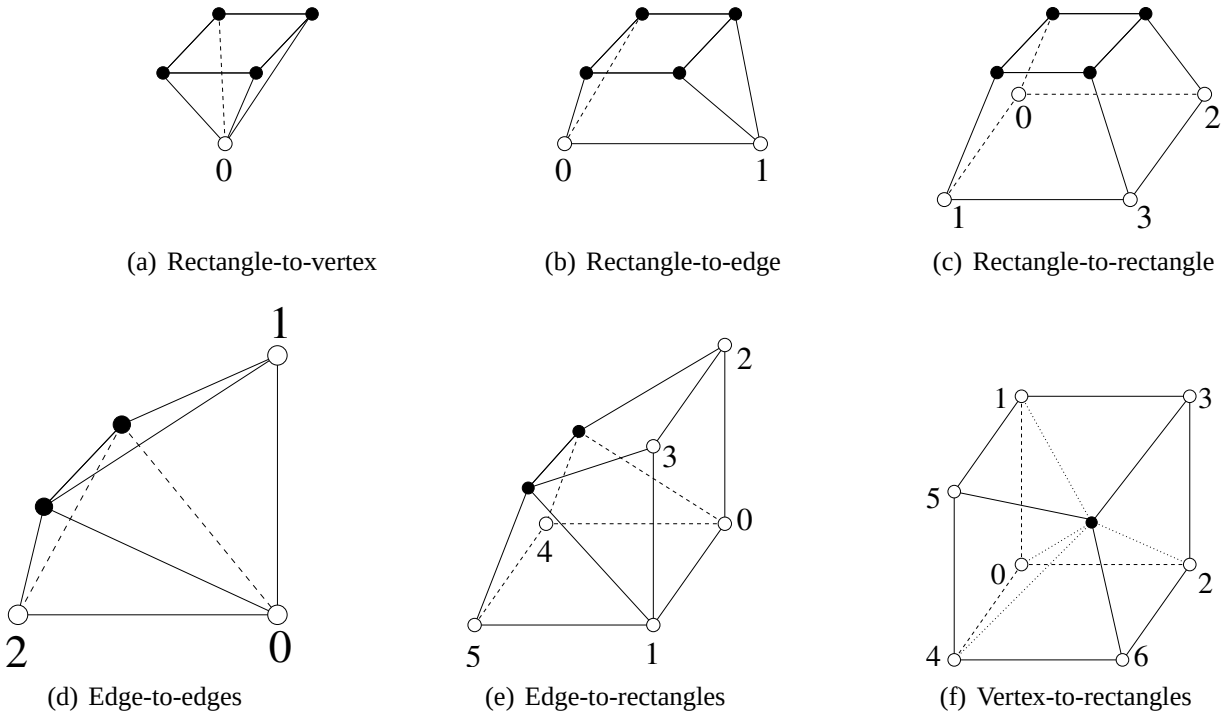
4.3 Stitching 3-d Grids

The index-based approach can be generalized to 3-d AMR grids. In the simple case of one fine grid embedded in a coarse grid, boundary faces, edges and vertices of the fine grid must be connected to the coarse grid.

Each of the six boundary faces of a grid consists of a number of rectangles defined by four adjacent grid points on the face. A boundary face is connected to the coarse mesh by connecting each of its comprising rectangles to the coarse grid. For each quadrilateral, the level indices of the four grid points that would extend the grid in normal direction are computed. These are transformed into level indices of the parent level by dividing them by the refinement ratio r_l of the fine level. In each of the two directions implied by a rectangle, these transformed points may have the same level index component. If they have the same index for a direction, the fine rectangle must be connected to one vertex in this direction; otherwise, it must be connected to an edge in that direction. The result is the same as the combination of two 2-d edge cases. The various combinations result in rectangles being connected to either a vertex, a line segment (in the two possible directions) or another rectangle. The cell types resulting from these connections are pyramids, see Figure 4.5(a), deformed triangle prisms, see Figure 4.5(b), and deformed hexahedral cells, see Figure 4.5(c).

An edge is connected to the coarse grid by connecting its comprising edge segments to the grids in the parent level. For each segment, the level indices of the six grid points that would extend the grid beyond the edge are computed. These indices are also transformed into level indices of the parent level. Depending on whether the edge segment crosses a boundary face of the original AMR grid or not, the edge must either be connected to three perpendicular edges or two rectangles of the coarse grid. This is equivalent to a combination of the vertex and edge connection types of the 2-d case. If the viewing direction is parallel to the edge segment (such that it appears to the viewer as a point), it must always be connected to two perpendicular edges of the coarse grid. In the direction along the edge, one connects it to a point or a parallel edge. Connecting an edge segment to the coarse grid results in two tetrahedra, shown in Figure 4.5(d), or two deformed triangle prisms, shown in Figure 4.5(e), as connecting cells.

A vertex is connected by calculating the level indices of the seven points that would extend the grid. These are transformed into level indices of the parent level. The result is the same as the



(a) Rectangle-to-vertex (b) Rectangle-to-edge (c) Rectangle-to-rectangle

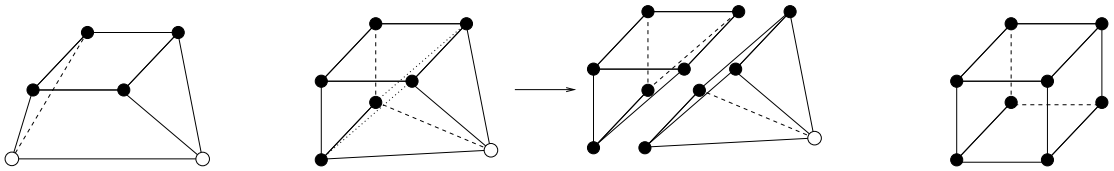
(d) Edge-to-edges (e) Edge-to-rectangles (f) Vertex-to-rectangles

Figure 4.5: Possible connection types for quadrilateral, edge and vertex in 3-d case

combination of two 2-d vertex cases. The vertex is connected to three rectangles of the coarse grid via pyramid cells, as shown in Figure 4.5(f).

When the coarse level is refined by more than one fine grid, one must check each coarse-grid point for refinement and adapt the generated tessellation accordingly. The simplest case is given when a fine-grid boundary rectangle is connected to a coarse-grid point, see Figure 4.5(a). If this coarse-grid point is refined, an adjacent fine grid exists and the fine-grid boundary rectangle must be connected to the other fine grid's boundary rectangle. This case illustrates that it is helpful to retain the indices within the fine level in addition to converting them to the coarse level. In the unrefined case, the rectangle was connected to a coarse point, because transforming the four level indices from the fine level to the coarse level yielded the same coarse-level index. If the grid point corresponding to that coarse level-index is refined, the "correct" fine boundary rectangle can be determined using the original fine level indices. (For a refinement ratio of $r_l = 2$ the correct choice of the fine-level rectangle is also implied by the connection type, but for general refinement ratios the fine-level index must be retained.)

Connecting a fine rectangle to a coarse edge, see Figure 4.5(b), is slightly more difficult. Each of the endpoints of a coarse edge can either be refined or unrefined. The resulting refinement configurations and their tessellations are shown in Figure 4.6. Refinement configurations, the cases in Figure 4.6 and subsequent figures are numbered as follows: For each connection type shown in Figure 4.5, the coarse-grid points that are connected to a fine grid element are numbered according to the corresponding sub-figure. A case number is obtained by starting with case 0.



(a) Case 0 (b) Cases 1 and 2 (c) Case 3

Figure 4.6: Refinement configurations for connecting a fine-grid rectangle to a coarse-grid edge

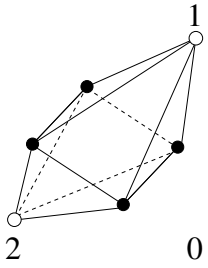
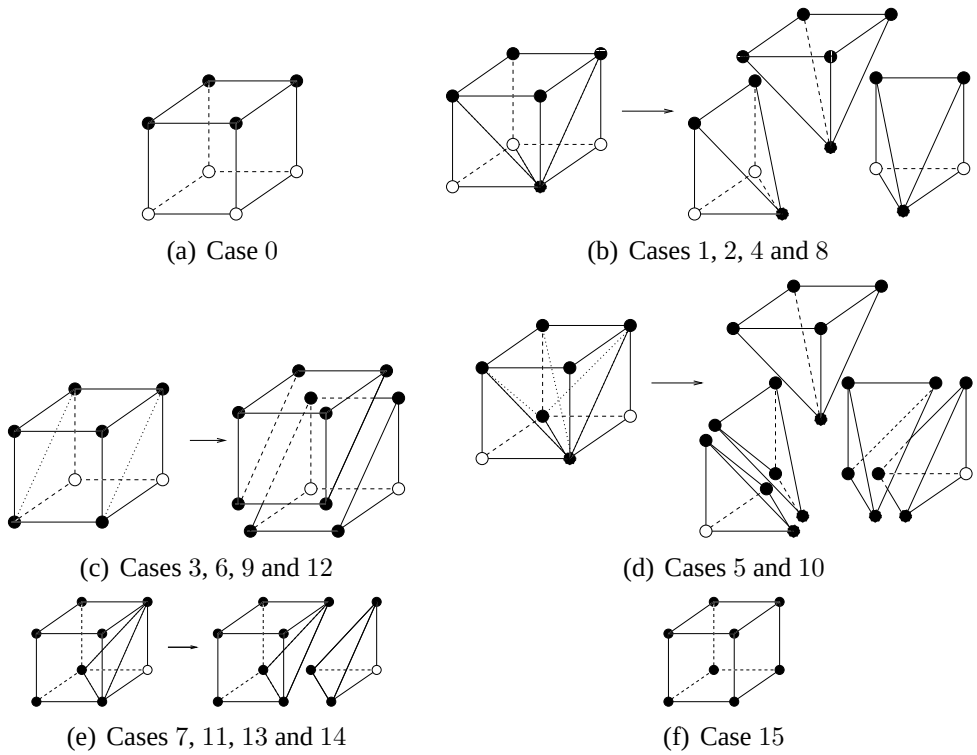


Figure 4.7: If coarse-grid vertex 0 is refined, when a fine-grid edge is connected to two coarse-grid edges, see Figure 4.5(d), two pyramids are generated instead of two tetrahedra

For each refined coarse-grid vertex k, 2^k is added to the number associated with this case. To determine to which refining grid points the fine rectangle should be connected, the fine-level indices are retained in addition to converting them to coarse level indices.

When connecting a fine-grid edge to two perpendicular coarse-grid edges, eight refinement configurations arise. If all coarse-grid vertices are unrefined, two tetrahedra are generated, as illustrated in Figure 4.5(d). If either coarse-grid vertex 1 or 2 is refined, the fine-grid edge must be upgraded to a coarse-grid rectangle and the corresponding tessellation function called with appropriate vertex ordering. (This procedure ensures that adjacent fine grids produce the same stitch tessellation as a continuous fine grid.) In the remaining case, when only coarse-grid vertex 0 is refined, two pyramids are generated instead of tetrahedra, see Figure 4.7.

For connection types where a fine-grid quadrilateral, see Figure 4.5(c), edge, see Figure 4.5(e), or vertex, see Figure 4.5(e), is connected to coarse-grid rectangles, eight points are considered. These points form a deformed hexahedral cell. One must consider 16 possible refinement configurations when a fine-grid rectangle is connected, since the four vertices belonging to the fine-grid rectangle are always refined, see Figure 4.8. More cases arise when a fine-grid edge or a vertex is connected to the coarse level. It is important to devise an efficient scheme to determine the tessellation for a given refinement configuration. Each cell face corresponds to a possible 2-d refinement configuration as shown in Figure 4.4. It is important to note that the 2-d refinement configurations that produce subdivided quadrilaterals are the same configurations that yield non-planar cell boundaries, *i.e.*, boundaries that must be subdivided. The subdivision information alone is sufficient to determine a tessellation for any given refinement configuration. It is



(a) Case 0 (b) Cases 1, 2, 4 and 8

(c) Cases 3, 6, 9 and 12 (d) Cases 5 and 10

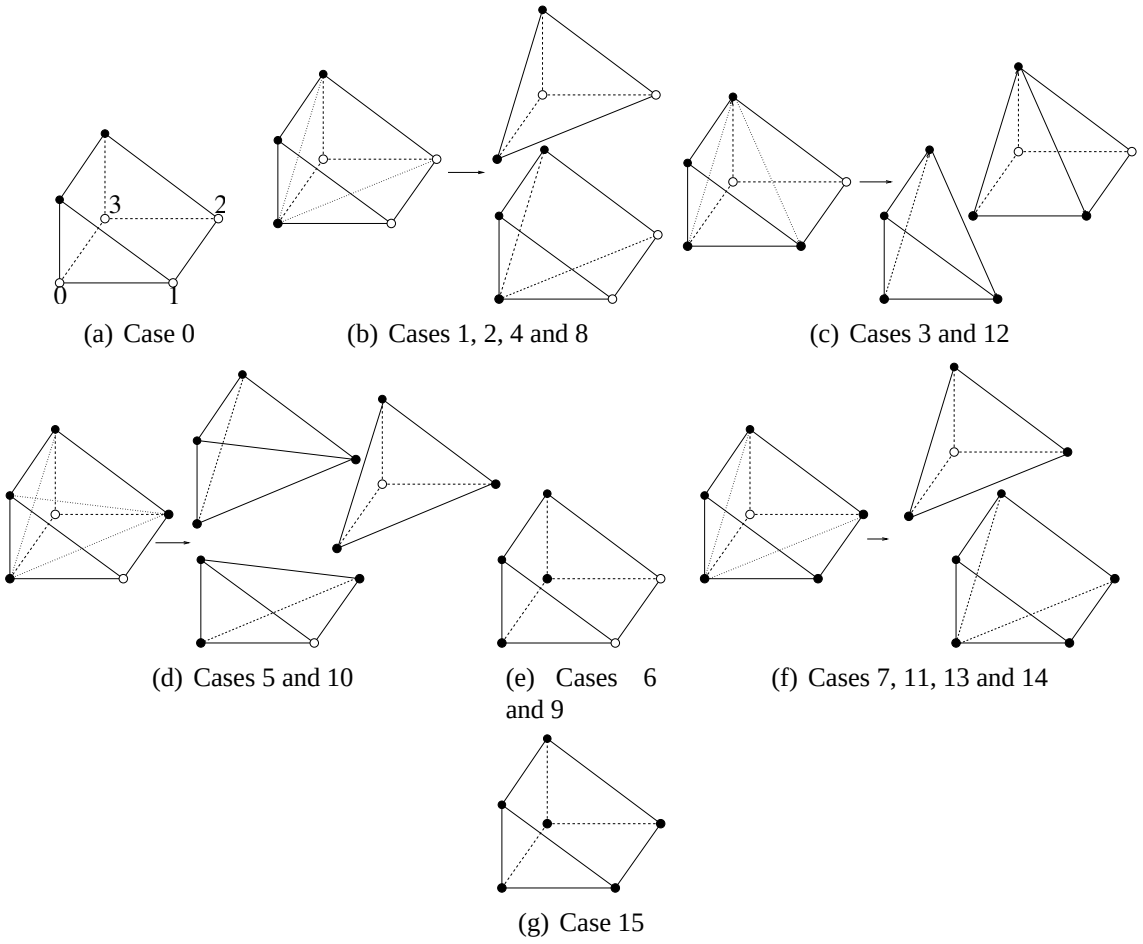
(e) Cases 7, 11, 13 and 14 (f) Case 15

Figure 4.8: Refinement configurations for connecting a fine-grid rectangle to a coarse-grid rectangle

not necessary to consider the positions of the points. Each cell face Figure 4.8 and subsequent figures is subdivided using the canonical tessellations depicted in Figure 4.4, illustrated by the dotted lines in a figure illustrating tessellations of a connection type, *e.g.*, Figure 4.8.

The subdivision of cell faces implies subdivisions of hexahedral cells into pyramids, triangular prisms and tetrahedra. In certain cases, see, for example, Figure 4.8(e), a cell type arises that does not correspond to the standard cells (hexahedra, pyramids, triangle prisms and tetrahedra), and that cannot be subdivided further without introducing additional vertices. Even though it is possible to generate a case table to extend MC to this cell type, the asymmetric form of this cell makes this extension difficult. Symmetry considerations that are used to reduce the number of cases that need to be considered cannot be applied. Therefore, cells of this type are handled by generating an additional vertex at the centroid of the cell. By connecting the centroid to all cell vertices one obtains a tessellation consisting of pyramids and tetrahedra.

Edges are connected to the coarse grid by considering eight vertices forming a deformed hexahedral cell. When an edge is connected, two of these eight points belong to the edge. The other six vertices are coarse-grid vertices and can be either refined or unrefined. Thus, it is necessary to consider 64 possible refinement configurations. It is necessary to consider all six coarse-grid points at once, since the boundary faces need not always be subdivided faces, as Figure 4.5(e) implies. If, for example, all coarse-grid points are refined, a single, not-tessellated



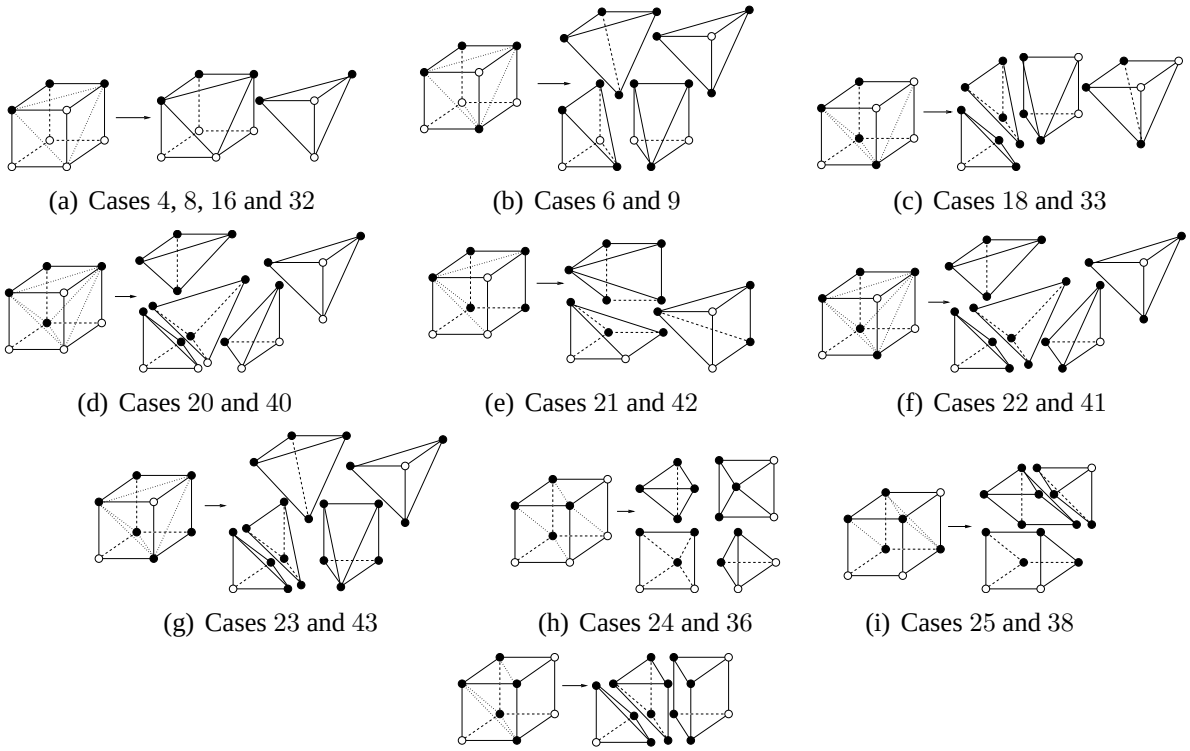
(a) Case 0 (b) Cases 1, 2, 4 and 8 (c) Cases 3 and 12

(d) Cases 5 and 10 (e) Cases 6 and 9 (f) Cases 7, 11, 13 and 14

(g) Case 15

Figure 4.9: Possible refinement configurations and corresponding tessellations for triangle prism cell

cell must be produced. However, in certain refinement situations (Cases 0–3, 5, 7, 10, 11, 15, 17, 19, 27, 34, 35, 39, 51), the two cell faces perpendicular to the fine-grid edge segment must be divided as shown in Figure 4.5(e). In these cases, it is possible to connect the fine-grid edge segment to the coarse grid by handling each of the triangle prisms separately. Each triangle prism is tessellated according to the refinement configurations shown in Figure 4.9. When connecting a fine-grid edge to two coarse-grid rectangles, see Figure 4.5(c), it must be upgraded to the rectangle case if either grid points 2 and 3 or grid points 4 and 5 are refined. In these cases (Cases 12–15, 28–31 and 44–63) the refinement configurations for connecting a fine-grid rectangle are used according to Figure 4.8. The tessellations for the remaining cases are shown in Figure 4.10. In Figure 4.10, I only considered symmetry with respect to a plane perpendicular to the edge to reduce the number of cases. The cases in Figures 4.10(i) and 4.10(j) differ only by rotation but yield the same tessellation. It is important to note, that only the partition of the boundary faces matters in determining a valid tessellation. Even though the refinement configurations shown



(a) Cases 4, 8, 16 and 32 (b) Cases 6 and 9 (c) Cases 18 and 33

(d) Cases 20 and 40 (e) Cases 21 and 42 (f) Cases 22 and 41

(g) Cases 23 and 43 (h) Cases 24 and 36 (i) Cases 25 and 38

(j) Cases 26 and 37

Figure 4.10: Remaining refinement configurations for connecting fine-grid edge to coarse-grid rectangles

Figures 4.10(d) and 4.10(f) differ, they yield the same partition of a cell's boundary faces and, thus, the same tessellation.

Vertices can be upgraded to edges, or even quadrilaterals, when more than two grids meet at a given location. The fine vertex shown in Figure 4.5(f) can be changed to an edge, if any of the coarse grid points 3, 5 or 6 is refined. In these cases, the procedure used to connect an edge segment is called with appropriate vertex ordering. As a result, the same tessellations are used as in the case of a continuing edge. Furthermore, this procedure ensures that an additional upgrade to the rectangle case is handled automatically when needed. In the remaining cases, each of the pyramids of the unrefined case can be handled independently. If the base face of a pyramid is not planar, it is subdivided using the corresponding configuration from Figure 4.4, and the pyramid is split into two tetrahedra.

4.4 Isosurface Extraction

Within individual grids, a slightly modified MC approach is used for isosurface extraction. Instead of considering all cells of a grid for isosurface generation, only those cells are considered

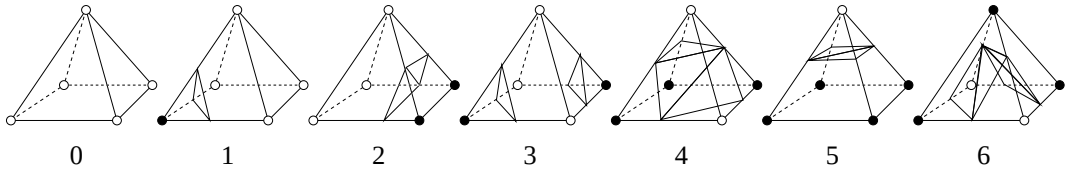


Figure 4.11: Basis configurations for triangulating a pyramid.

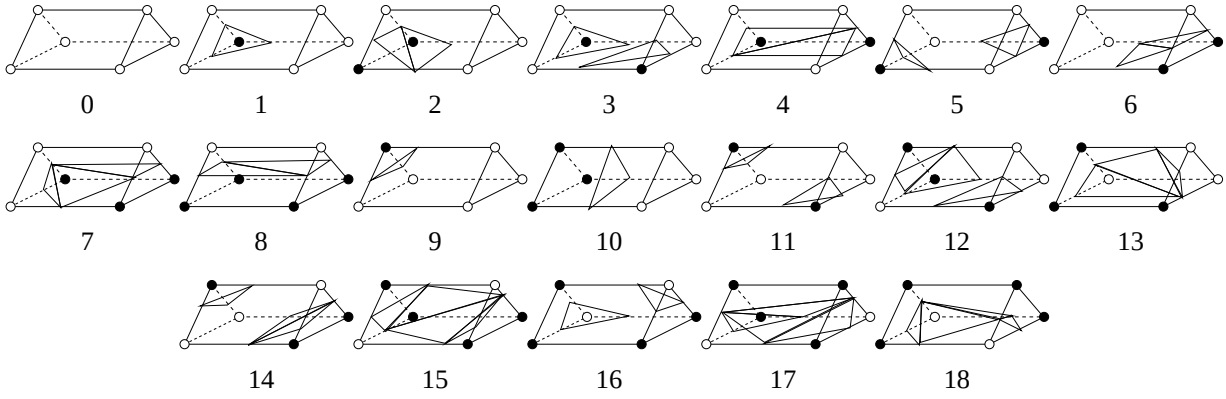
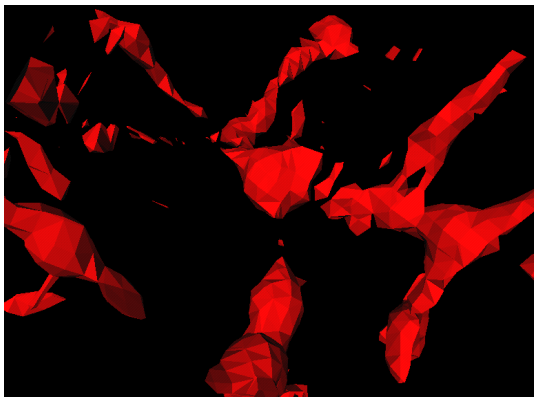


Figure 4.12: Basis configurations for triangulating a triangle prism.

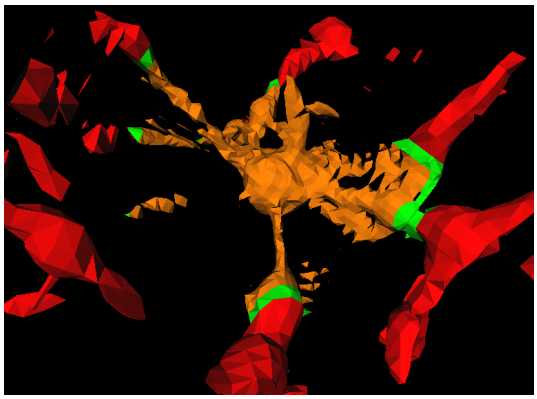
that are not refined by a finer grid. This is done by pre-computing a map with refinement information. For each grid cell, this map contains an index of a refining grid or an entry indicating that the cell is unrefined. This makes it possible to quickly skip refined portions of the grid. For the generation of an isosurface within stitch cells, the MC method must be extended to handle the cell types generated during the stitching process. This extension is achieved by generating case tables for each of the additional cell types. These new case tables must be compatible with the one used in the standard MC approach. For rectilinear cells, the case table from VTK [74] is used, which is based on implicit disambiguation. (On ambiguous faces, positive vertices are always separated.) By defining triangulations for tetrahedral cells, pyramid cells, see Figure 4.11, and triangle prism cells, see Figure 4.12, crack-free isosurfaces can be extracted from an AMR data set. For all cell types, implicit disambiguation is used, consistently separating positive vertices. Pyramids and triangle prisms are not rotational symmetric. To reduce the number of base cases, one can use symmetry concerning mirroring a configuration at faces perpendicular to the x- and z-axis, subdividing a cell along its center in that direction. When a configuration contains no ambiguous face, inversion can be used to further reduce the number of basis configurations.

4.5 Results

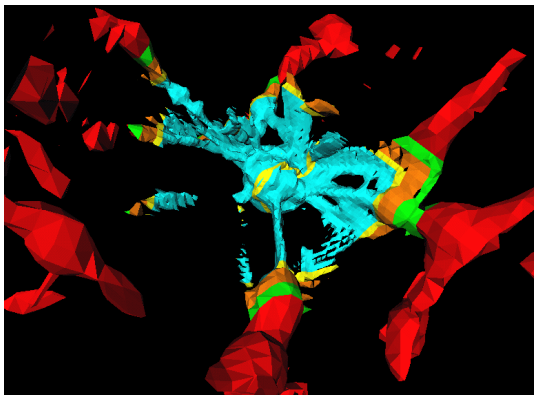
Figure 4.13 shows isosurfaces extracted from an AMR data set. This data set is the result from an astrophysical simulation of star clusters performed by Bryan [8]. Figure 4.13(b) shows an isosurface extracted from two levels of the hierarchy, and Figure 4.13(c) one extracted from three levels. To highlight the transitions between levels, the parts of the isosurface extracted from different levels of the hierarchy are colored differently. Isosurface parts, extracted from the root, the first and the second levels, are colored red, orange and light blue, respectively. Portions extracted from the stitch meshes between the root and the first level are colored in green, and portions extracted from the stitch meshes between the first and second levels are colored in yellow. The root level and the first level of the AMR hierarchy each consist of one $32 \times 32 \times 32$ grid. The second level consists of 12 grids of resolutions $6 \times 12 \times 6$, $6 \times 4 \times 2$, $8 \times 12 \times 10$, $6 \times 4 \times 4$, $14 \times 4 \times 10$, $6 \times 6 \times 12$, $12 \times 10 \times 12$, $10 \times 4 \times 8$, $6 \times 6 \times 2$, $16 \times 26 \times 52$, $14 \times 16 \times 12$ and $36 \times 52 \times 36$. All measurements were performed on a standard PC with a $700\mathrm{MHz}$ Pentium III processor.



(a)



(b) Stitch cell generation: approximately 55ms; isosurface generation: approximately 250ms



(c) Stitch cell generation required approximately 340ms; isosurface generation: approximately 600ms

Figure 4.13: Isosurface extracted from AMR hierarchy simulating star clusters using only the root level (a), two of seven levels (b) and three of seven levels of an AMR hierarchy (c). (Data set courtesy of Greg Bryan, Massachusetts Institute of Technology, Theoretical Cosmology Group, Cambridge, Massachusetts)

Chapter 5

Direct Volume Rendering of Adaptive Mesh Refinement Data

5.1 Hardware-accelerated Volume Rendering of AMR Data

5.1.1 Introduction

Visualizations based on direct volume rendering require the specification of meaningful viewpoints as well as transfer functions emphasizing features in a data set. Even though approaches for the automatic creation of transfer functions are known, there is still a need for generating customized transfer functions. Finding a "good" transfer function is often done on a trial-and-error basis. It is desirable to have a means for visualizing a volumetric data set at interactive speed as this greatly enhances a users understanding of data. Therefore, I have adopted hardware-accelerated volume-rendering schemes for AMR data.

5.1.2 Rendering a Single Grid

Instead of using a hardware-accelerated volume-rendering approach based on texture hardware, see Section 2.2.4, cell projection is simulated using graphics hardware. This approach is restricted to orthographic projection, ensuring that front-facing and back-facing faces are the same for all cells. When dealing with more general grid cells, each boundary face must be checked individually whether it is back-facing or front-facing. This step can be done, for example, by using the scalar product between face normal and a vector $\mathbf{tv}$ (a vector directed toward the viewer, see Figure 5.1(a)). For axis-aligned rectilinear grids, this test can be performed based on the viewing direction. Figure 5.1 shows the numbering of the faces of a cell of a rectilinear grid, and Table 5.1 lists the criteria used to determine whether a face is front- or back-facing. Front- and back-facing faces are the same for each cell in all grids. It is sufficient to determine front-facing faces once.

Within cells, constant interpolation is used. A value associated with the cell is assigned to all locations within the cell. Using a transfer function, corresponding color and opacity values for the cell are determined. The set of front-facing faces defines a "footprint" of cells. The

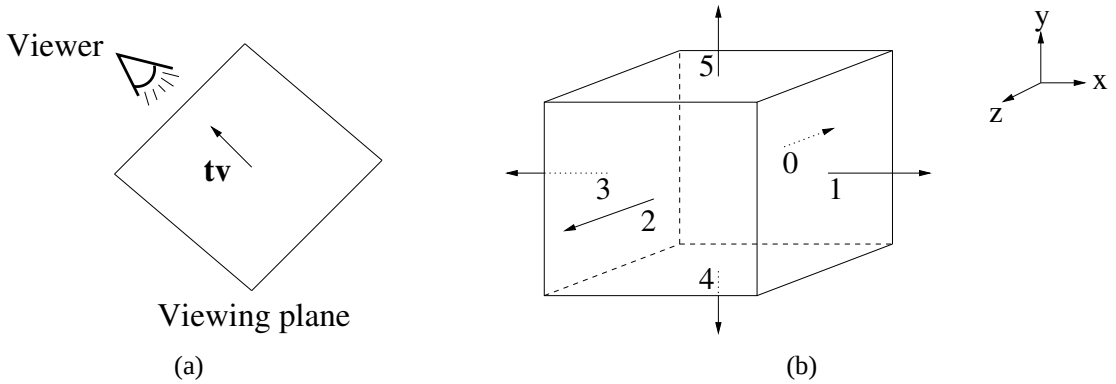


Figure 5.1: Determining front- and back-facing faces. (a) The vector $\mathbf{tv}$ is perpendicular to the viewing plane, pointing toward the viewer. (b) Face numbering used in Table 5.1.

Face #	Front-facing	Back-facing	Perpendicular
0	$tv_z < 0$	$tv_z > 0$	$tv_z = 0$
1	$tv_x > 0$	$tv_x < 0$	$tv_x = 0$
2	$tv_z > 0$	$tv_z < 0$	$tv_z = 0$
3	$tv_x < 0$	$tv_x > 0$	$tv_x = 0$
4	$tv_y < 0$	$tv_y > 0$	$tv_y = 0$
5	$tv_y > 0$	$tv_y < 0$	$tv_y = 0$

Table 5.1: Criteria used for checking whether a cell face is front-facing, back-facing, or should not be rendered.

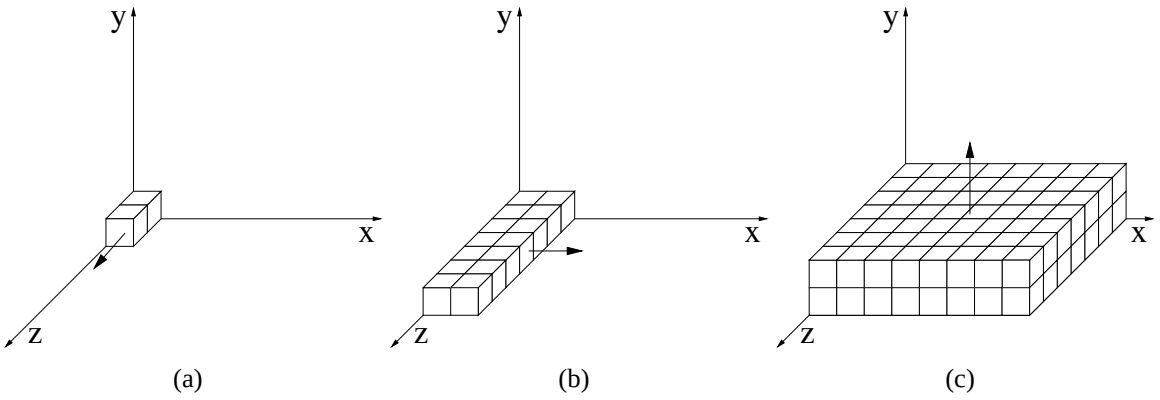


Figure 5.2: Rendering order of grid cells — all components of $\mathbf{tv}$ being positive. The order in which axes are handled (first-z-then-x-then-y order) is arbitrary. Only the order according to which cells are handled along an axis is important. This order is determined based on the corresponding component of $\mathbf{tv}$.

cell footprint is rendered by displaying all front-facing faces of the current cell with constant color and opacity. This is similar to splatting, see Westover [89], using a convolution kernel that assigns a constant value to all positions within the cell outline. Cells are rendered in back-to-front order using three nested loops, one loop for each axis. The order according to which axes are handled is arbitrary. Along each axis, cells must be handled in correct order. This order can be determined based on the sign of the component of the vector $\mathbf{tv}$ corresponding to the axis handled by the loop. If it is positive, cells are enumerated in ascending axis direction. If it is negative, cells are enumerated in descending axis direction. If it is zero, an arbitrary choice is made. Figure 5.2 shows the rendering order of cells when all components of $\mathbf{tv}$ are positive.

In order to account for differing cell lengths in different AMR levels, a transfer function specifies opacity per unit length. Before rendering a grid, a transformed transfer function is computed and subsequently used for color and opacity lookup during rendering of front-facing faces. This transformed transfer function specifies colors and opacities for a ray segment with the length of a cell viewed along the coordinate axis "most parallel" to the viewing direction. First, the individual components of the viewing vector $\mathbf{tv}$ are considered, and the axis corresponding to the component with the largest absolute value detemined. Subsequently, the cell size along that direction is used as cell length l for the transformed transfer function. The transformed opacity is computed as

$$\alpha_{\text{Transformed}} = 1 - (1 - \alpha)^l \ . \tag{5.1}$$

(The color component is the same as in the original transfer function since color in the transfer function are specified without pre-multiplied alpha value.) Rendering all front-facing faces with constant color ignores the fact that the length of ray-segments in a cell generally varies with their location and produces artifacts in addition to the "blockyness" of constant interpolation. This is illustrated in Figure 5.3. When a cell is viewed in axis direction, all rays traverse the same distance in the cell (equal to the length of the cell in that direction), shown in Figure 5.3(a). When the cell is not viewed in an axis direction, the length of the segment of the ray within this cell depends on the position of the ray. The quick-preview method ignores this fact. It is important to note that the resulting artifacts are similar to the artifacts produced by a slicing approach that uses axis-perpendicular planes to composite volume images. If an image rendered by this method is viewed in the direction of an axis, the results are "correct." The more a viewing direction deviates from an axis direction, the more the individual planes are visible. The resulting renderings are similar to those one would obtain when rendering all three sets of planes at the same time rather than choosing the set that is most perpendicular to the viewing direction. This results in slightly different artifacts. The cell based-approach allows one to render only those cells whose opacity is above a given threshold and generate less geometry.

5.1.3 AMR Partition Tree

Structured-rectilinear grids allow the use of hardware-based schemes to accelerate rendering, and AMR data consists of structured-rectilinear grids. However, when viewed over the whole domain there are changes in resolution. In order to apply hardware-accelerated schemes to AMR data, I partition a given data set into blocks of constant resolution using a generalized k-d tree [5].

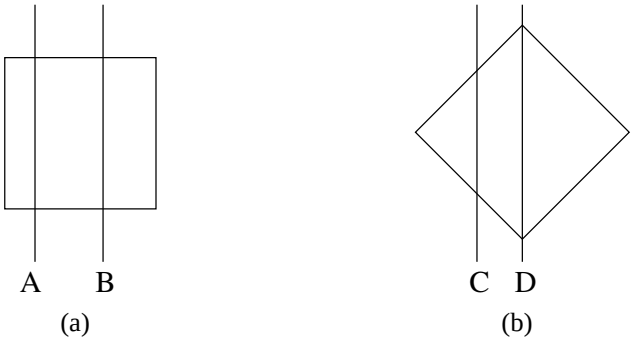


Figure 5.3: Different lengths of ray traversing cell. (a) Rays axis-aligned. (b) Rays not axis-aligned.

Each grid is partitioned into two types of blocks of adjacent cells: A block consists completely of cells for which no refined representation is available or all cells of the given block are available in a higher resolution. In the latter case, the block contains a pointer to the finer representation. The resulting blocks are rendered from back to front. During rendering, it is determined for each block whether it is rendered at low or high resolution to achieve interactive frame rates.

The partition tree consists of nodes of different types. All nodes share certain common information, regardless of type. Each node specifies a region of an AMR grid. Thus, each node contains at least a reference to this AMR grid and information regarding which cells of the grid are used by this node. These regions are always rectilinear and axis-aligned, allowing their specification by the minimum and maximum cell indices. All cells with indices between these indices belong to the region described by this node. This information alone is sufficient for the "unrefined-grid-part" node. Nodes of this type describe regions in an AMR grid for which no further refinement information is available, regions that can always be rendered at the fixed resolution of the current level. The "completely-refined-grid-part" node describes a region that is completely available at a higher resolution and thus indicates a transition between two hierarchy levels. The corresponding region of the AMR grid is completely available at a higher resolution. Consequently, it is possible to render the resolution of the current level or alternatively render it at a higher resolution. This higher-resolution representation is a pointer to a sub-tree describing that region. A "partition" node splits a given region into "stripes" along an axis. Each stripe is an unrefined-grid-part node, a completely-refined-grid-part node, or a partition node (with a different partition direction).

Figure 5.4 shows the partition tree for the AMR hierarchy from Figure 2.47. To render the complete hierarchy, this partition tree is traversed. Partition nodes are handled by rendering their children in correct order, *i.e.,* by traversing their children lists back-to-front. (I use orthographic projection.) Thus, back-to-front traversal can be achieved by checking the component of the view-direction vector corresponding to the partition direction and rendering the child nodes in a direction according to the sign of the component. Unrefined-grid-part nodes are handled by rendering the corresponding region of the AMR grid using a hardware-accelerated approach. Completely-refined-grid-part nodes are rendered by traversing their sub-trees.

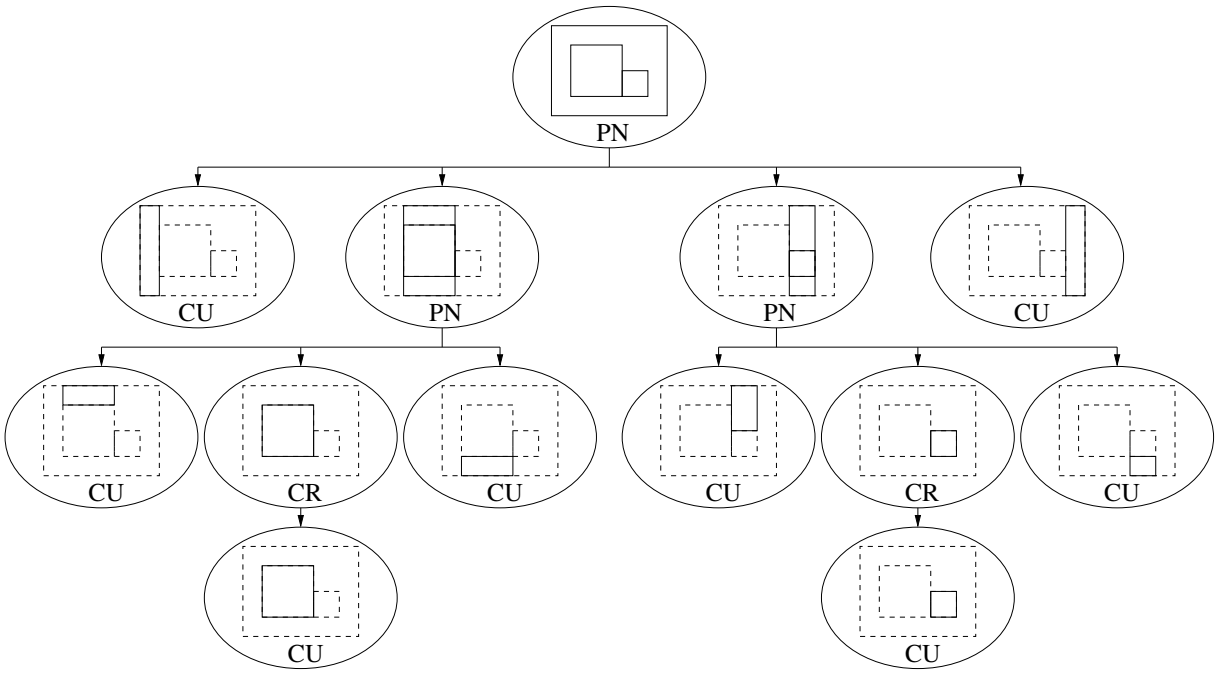


Figure 5.4: Partition tree for hierarchy shown in Figure 2.47. Completely refined grid-part nodes are denoted by "CP," unrefined grid-part nodes are denoted by "CU," and partition nodes are denoted by "PN."

.

5.1.4 Adaptive Tree Traversal

By using heuristics it is possible to adaptively traverse the partition tree and render images that take certain criteria into account. Two types of criteria governing the traversal of the partition tree exist: *viewpoint-dependent* criteria aim at minimizing rendering time while retaining quality of generated images; *time-constraining* criteria aim at maintaining interactive rendering rate, sacrificing image quality if necessary. Two viewpoint-dependent criteria to modify tree traversal are employed. During the traversal of a partition node, a check is performed for each child whether the region corresponding to that node is visible, *i.e.*, whether it intersects the viewing frustum. If the answer is negative, the corresponding child is skipped, thus culling the corresponding part of the volume. Since an intersection test for bounding box and viewing frustum is too expensive to be performed in real time, a heuristic is used: An arbitrary diagonal of the region (the line segment spanning from one vertex of a 3-d cell to the opposite-corner vertex) is projected to view coordinates and a check performed whether the bounding box of that diagonal intersects the view region. (In normalized view coordinates this region is given as $[-1,1] \times [-1,1] \times [-1,1]$.) When a completely refined-grid-part node is traversed, a choice must be made: Should the corresponding region be rendered at the resolution of the current level or should it be rendered at a higher resolution? The decision is based on an estimated footprint of the voxel. If each cell of a refining grid covers only sub-pixels, *i.e.*, only fractions of pixels, it is sufficient to render the data with

the resolution of the parent level. Again, to perform this decision efficiently enough for real-time rendering this region is estimated by using the bounding box of the projected diagonal of a cell.

Considering viewpoint-dependent criteria alone only avoids unnecessary rendering time but does not necessarily ensure interactive rendering speed. When it is not possible to use the full resolution of an AMR data set and achieve interactive frame rates, greater speed can be obtained by sacrificing rendering quality. Currently, a traversal depth of the partition tree for interactive rendering is specified. For each frame, the time necessary to render it is measured, and an achieved frame rate estimated. If the estimated frame rate is too low, the traversal depth is decreased; if it is higher than the desired frame rate, the traversal depth, and thus the quality of the rendered image, is increased.

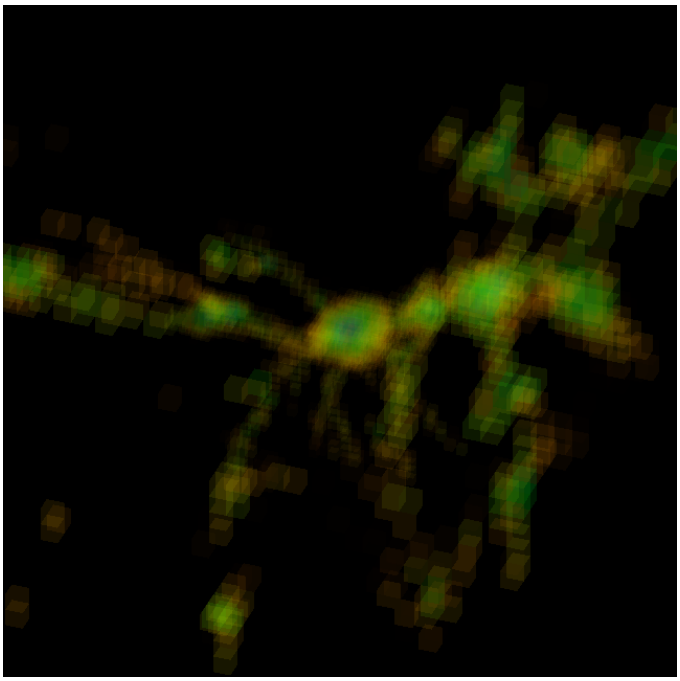
5.1.5 Results

Figure 5.5 shows images used for interactive preview purposes for AMR data. The considered AMR data set is the same astrophysical data set used in Section 4.5. On an SGI Onyx with InfiniteReality2 graphics, a frame rate of five frames per second can be achieved while using almost the entire AMR hierarchy (five to six of eight levels). Even on an Indigo2 with SolidImpact graphics four to five of eight levels can be rendered with five frames per second, since my method requires no hardware acceleration of textures. Using an NVIDIA GeForce board, only two to three levels can be rendered at five frames per second. The result is shown in Figure 5.5(a). For merely choosing a viewpoint, this quality is sufficient. The best possible quality using the hardware-accelerated approach is shown in Figure 5.5(b). This image utilizes all levels in the AMR hierarchy and uses the view-dependent criteria merely to avoid unnecessary computation time. This image requires about two seconds on an SGI Onyx with InfinityReality2 graphics and about ten seconds on a PC with NVIDIA GeForce board.

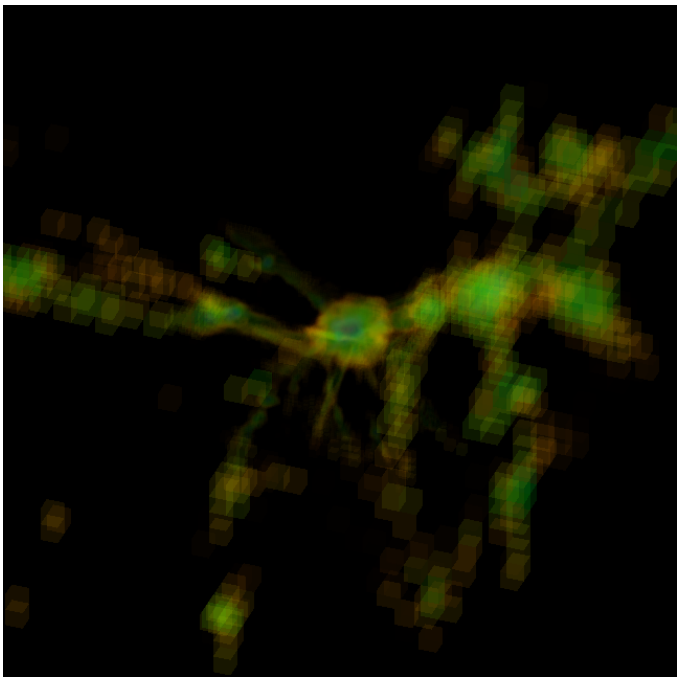
5.2 Progressive high-quality rendering using cell projection

5.2.1 Introduction

After choosing a suitable viewpoint and transfer function, it becomes necessary to render higher-quality images. I have developed an approach based on cell projection, that utilizes different interpolation schemes. In some cases, *e.g.*, for debugging purposes, it is desirable to use only data values that were computed in an AMR simulation. For these purposes, constant interpolation which assigns the data value associated with a cell to all locations within that cell, is most suitable. To improve the quality of rendered images, I examined the suitability of the piecewise linear method (PLM), one of the interpolation schemes used during simulation, for visualization purposes. PLM is not C^0-continuous, resulting in visible artifacts in rendered images. This problem can be resolved by using trilinear interpolation and dual grids to define a C^0-continuous, consistent interpolation scheme that can be used to obtain high-quality images. My cell-projection-based rendering scheme allows for progressive rendering of an AMR hierarchy. Constant interpolation and PLM operate on original grids of an AMR hierarchy. My scheme al-



(a)



(b)

Figure 5.5: Images generated by hardware-accelerated volume renderer. (Data set courtesy of Greg Bryan, Massachusetts Institute of Technology, Theoretical Cosmology Group, Cambridge, Massachusetts) (a) Data set rendered during user interaction based on two coarsest-level grids in AMR hierarchy. (b) Data set rendered at full resolution.

lows one to refine a rendered image grid-by-grid when using these interpolation schemes. When using dual grids, it is not possible to refine on a per-grid basis. Consequently, I modify my approach to allow refinement on a per-level basis.

5.2.2 Progressive Refinement Cell Projection of AMR Data

An AMR hierarchy is rendered by subsequently cell-projecting its constituent grids using the ray-segment-queue-based cell-projection approach introduced by Ma and Crocket [58], see Section 2.2.4. Two schemes are possible: Bottom-up rendering starts with rendering the finer grids and proceeds with filling gaps by rendering the corresponding portions of the coarser grids. In a top-down approach, a coarse grid is rendered first. The result can be displayed and used as an intermediate visualization. The rendered image is refined by proceeding to the finer grids and replacing portions of the already rendered image with a version at higher resolution. A simple bottom-up rendering scheme starts by cell-projecting the grids of the finest level. Subsequently, grids of the coarser levels are cell-projected. While rendering these coarser grids, special care must be taken of those cells overlapped by a finer grid. Ray segments for the regions covered by these cells are already computed. Thus, these cells must be skipped during the rendering process to obtain a correct result. This is done by using an *intersection map*. The intersection map contains an entry for each cell in the grid that specifies the index of the grid that overlaps this cell, should such a grid exist. During the rendering process, the intersection map entry for the current cell is read. If it specifies an overlapping grid, the cell is skipped. A straightforward extension is to render only a subset of refining grids. In this case, a cell is skipped only when the grid overlapping that cell is already rendered.

By adding supplemental information to each ray segment that is inserted into the ray-segment queues, it is possible to use a top-down rendering approach for AMR hierarchies. For each ray segment, indices of two grids are stored: One index specifies the grid in which the segment was created, *i.e.*, the grid to which the cell belongs. The other index specifies by which grid in the child level the segment is affected, *i.e.*, it specifies rendering which grid would define a more accurate representation of that ray segment. This is the index of the grid refining the cell in which the ray segment was created and can be obtained by reading the corresponding entry in the intersection map. Ray segments are only merged when they are adjacent (see Section 2.2.4) were created in the same and are affected by the same grid. Figure 5.6 shows three rays traced from a specific viewpoint. When the coarsest grid G_0 is rendered, all ray segments are tagged as being created in grid G_0. Ray A is not affected by any grid, since it does not intersect any refined grid cells. Ray B is divided into four segments: the first is not affected by another grid; the second is affected by grid G_1; the third is affected by grid G_2; and the fourth is not affected by any grid. Ray C consists of three segments: the first is not affected by another grid; the second is affected by grid G_1; and the third is not affected by another grid. After rendering grid G_0 completely, the ray-segment queue for ray A contains exactly one ray segment. The queues for rays B and C contain four and three segments, respectively. Ray B consists of a segment from the viewpoint to grid G_1 (solid line segment), one segment that is contained in grid G_1 (dashed line segment), a segment that is contained in grid G_2 (solid line segment), and a segment after leaving grid G_2 (dashed line segment). Ray C is comprised of a segment from the viewpoint to grid G_1 (solid

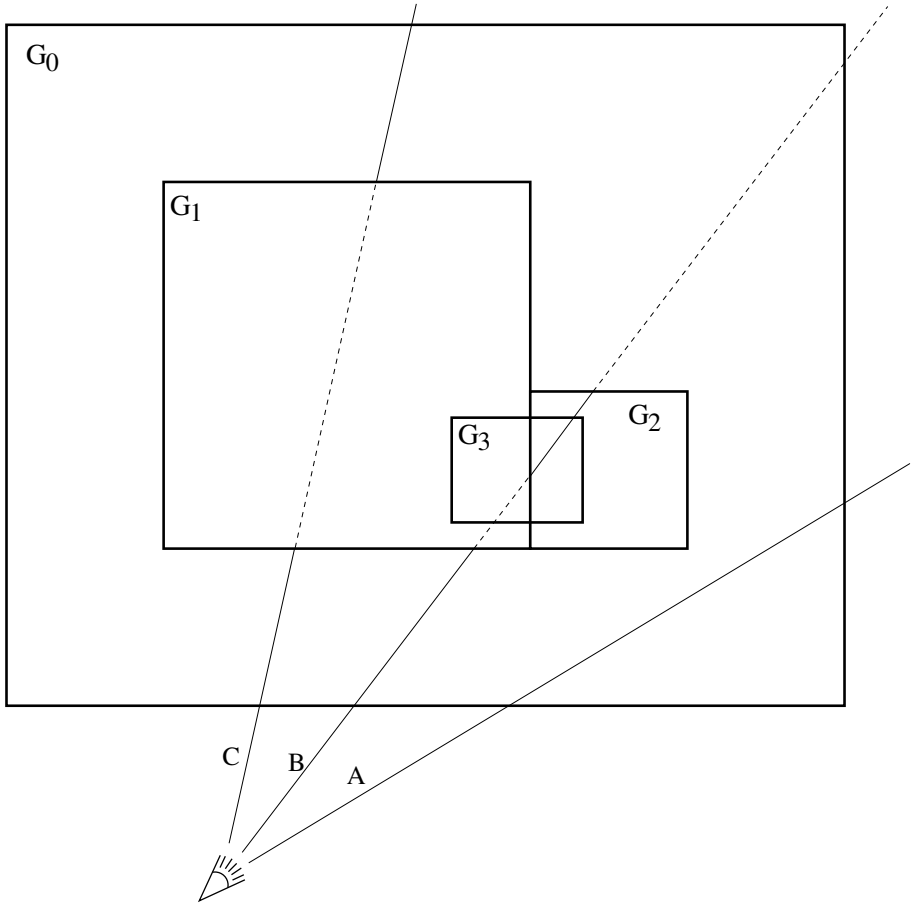


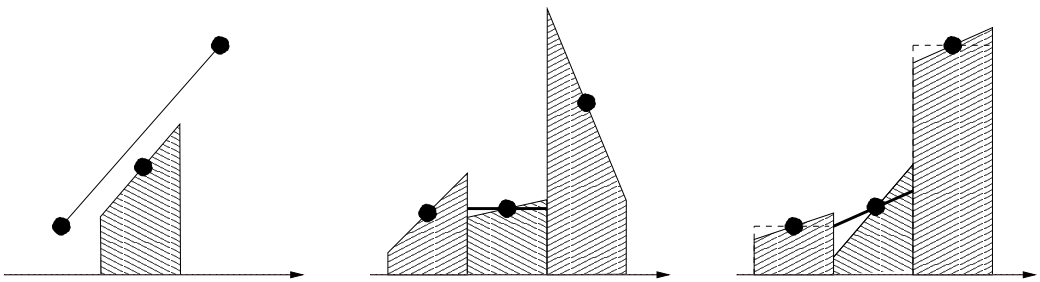
Figure 5.6: Progressive rendering of AMR hierarchy.

line segment), a segment that is contained in grid G_1 (dashed line segment), and the segment after leaving grid G_1 (solid line segment). Since ray segments can only be merged when they are affected by the same grid, these segments cannot be merged. It is important to note that, even though ray B intersects also grid G_3, this intersection is not considered until grids G_1 and G_2 are rendered. Only grids of the next-finer level are considered as affecting a given ray segment.

Using this approach of partitioning rays into segments that are affected by finer grids and those that are not, it is straightforward to refine an already rendered image by rendering a finer grid. Before the finer grid is rendered, all ray segments affected by that finer grid are erased from the ray-segment queues. When the finer grid is rendered, the gaps in the ray segments resulting from this step are filled with more accurate ray segments, resulting in an improved image.

5.2.3 Constant Interpolation and Piecewise-Linear Method

One way to render grids of an AMR data set is not to use interpolation at all and use only original data values. This approach assigns the value at a cell's center to the whole extent of



(a) (b) (c)

Figure 5.7: Computing interpolated values using PLM. (a) The first derivative is approximated as the difference between the values at the centers of the two adjacent cells divided by their distance. Values within the cell are interpolated linearly by adding the approximated derivative multiplied by the distance from the cell center to the value at the cell center. (b) If the cell value is a local extremum, the first derivative is set to zero (bold solid line) instead of using the difference between adjacent cell values (solid line). (c) If necessary, the absolute value of the approximated derivative (solid line) is reduced (heavy solid line) to restrict the values in a grid cell between the average levels (dashed lines) of its neighbors.

the cell. Scientists who perform simulations favor this approach, as it might make it easier to identify problems in a simulation.

If an interpolation scheme is used, it should be compatible with the interpolation scheme applied during the simulation. Unfortunately, this is complicated by the fact that the underlying finite-difference method has no "inherent" interpolation scheme. Differentiation is approximated by differencing function values of neighboring grid points. The simulation does not use any values besides those at cell centers. AMR, however, uses an interpolation scheme to initialize values of newly created grids. It is possible to use an interpolation scheme based on the piecewise-linear method (PLM), commonly used in AMR simulations, for visualization. PLM can be used with simulations of arbitrary dimension. In the following, the basic method is described for the 2-d case. PLM estimates the partial derivatives at the center of a grid cell $\mathbf{k} = (k_0, k_1)$ using the difference between grid-cell values, *i.e.*,

$$\left.\frac{\partial}{\partial x_0} f\right|_{\mathbf{x}=\mathbf{p_k}} = \frac{v_{(k_0+1,k_1)} - v_{(k_0-1,k_1)}}{2\delta_0} \quad \text{, and} \quad \left.\frac{\partial}{\partial x_1} f\right|_{\mathbf{x}=\mathbf{p_k}} = \frac{v_{(k_0,k_1+1)} - v_{(k_0,k_1-1)}}{2\delta_1} \,, \tag{5.2}$$

where δ_0 and δ_1 denote the cell size in the corresponding directions. Figure 5.7(a) illustrates this approximation for the 1-d case (considering one component). The scalar value associated with the cell center is denoted by $v_\mathbf{k}$ and its position, *i.e.*, the center of cell $\mathbf{k}$, denoted by $\mathbf{p_k}$. Scalar values in a cell are approximated linearly by adding the first derivative multiplied by the distance from the cell center to the scalar value at the cell's center:

$$\text{PLM}(\mathbf{x}) = v_\mathbf{k} + (\mathbf{x} - \mathbf{p})\nabla f\Big|_{\mathbf{x}=\mathbf{p_k}} = v_\mathbf{k} + (x_0 - p_0)\frac{\partial}{\partial x_0} f\Big|_{\mathbf{x}=\mathbf{p_k}} + (x_1 - p_1)\frac{\partial}{\partial x_1} f\Big|_{\mathbf{x}=\mathbf{p_k}} \,. \tag{5.3}$$

Using the first derivative according to Equation 5.2 can result in the introduction of new extrema. PLM prevents this from happening by imposing a limit on the absolute value of the first derivative ("van Leer Limiting [79]"). This is done by extending Equation 5.2 in the following way: First, a decision is made whether a local extremum is present at the cell center of the current cell. A local extremum in direction 0 is present when the two differences $v_{(k_0,k_1)} - v_{(k_0-1,k_1)}$ and $v_{(k_0+1,k_1)} - v_{(k_0,k_1)}$ have different signs, *i.e.*, when

$$\left(v_{(k_0,k_1)} - v_{(k_0-1,k_1)}\right)\left(v_{(k_0+1,k_1)} - v_{(k_0,k_1)}\right) < 0\ . \tag{5.4}$$

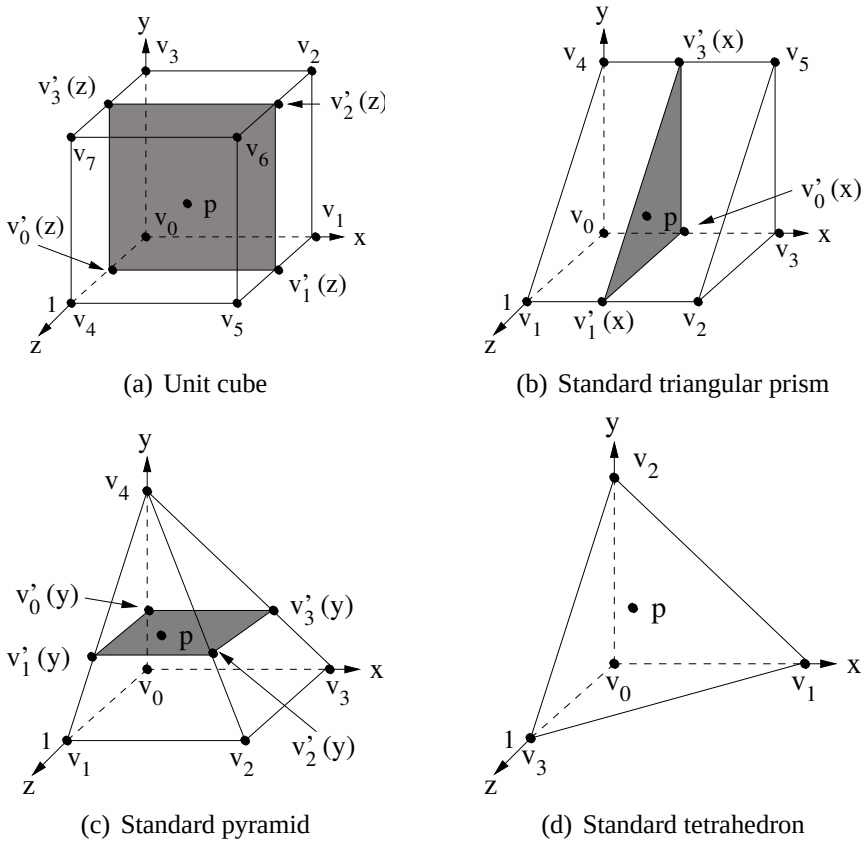
In this case, the corresponding partial derivative is set to zero, see Figure 5.7(b). Otherwise, the absolute value of the slope is set to

$$\begin{aligned}\left|\frac{\partial}{\partial x_0} f\Big|_{\mathbf{x}=\mathbf{p_k}}\right| = \min\Bigg(\Bigg\{ &\left|\frac{v_{(k_0+1,k_1)} - v_{(k_0-1,k_1)}}{2\delta_0}\right|, 2\left|\frac{v_{(k_0+1,k_1)} - v_{(k_0,k_1)}}{\delta_0}\right|, \\ &2\left|\frac{v_{(k_0,k_1)} - v_{(k_0-1,k_1)}}{\delta_0}\right|\Bigg\}\Bigg)\ .\end{aligned} \tag{5.5}$$

The last two terms ensure that interpolated values in a cell lie between the average values (*i.e.*, the values at the cell centers) of adjacent cells, see Figure 5.7(c). When the condition stated in Equation 5.4 holds, all three terms in the "min" expression have the same sign. The sign of the approximated derivative is set to this sign. The other direction, 1, is handled analogously. This approach generalizes to higher dimensions. (If the values in the 2-d case are interpreted as height values over the grid, the interpolant defines a plane with slopes $\frac{\partial}{\partial x_0} f|_{\mathbf{x}=\mathbf{p_k}}$ and $\frac{\partial}{\partial x_1} f|_{\mathbf{x}=\mathbf{p_k}}$ in x_0– and x_1–direction, respectively, passing through the scalar value at the cell center.)

PLM is well suited for interpolation of border values for a new grid. When used for visualization purposes, PLM can cause problems, since it is not continuous at cell boundaries. Even though for most viewing angles integration smooths out artifacts in volume renderings, distracting artifacts can still result when the view direction is along one of the three major axes and two neighboring rays pass through two different "stacks" of cells. Each cell intersected by the ray has a significantly different color value compared to the color value of the neighboring cells. This effect is accumulated by the integration along the ray.

When using constant interpolation or PLM an unmodified cell-projection approach can be used. Constant interpolation assigns a single value to all locations within a cell. When boundary faces are scan converted, this constant value is assigned to all pixels as interpolated value. (In fact, an optimized cell-projection approach that only deals with constant interpolation need not compute and store interpolated values for individual pixels.) All locations along a ray segment have the same value and optical properties. Consequently, it is possible to evaluate a particle light model analytically, see Section 2.2.4. For the piecewise linear model, all interpolated values lie on a 3-d hyperplane. Interpolated values can be obtained by linearly interpolating within boundary polygons and subsequently interpolating along a ray segment. This is equivalent to the standard cell-projection approach for linear tetrahedra.



(a) Unit cube (b) Standard triangular prism

(c) Standard pyramid (d) Standard tetrahedron

Figure 5.8: Base elements for interpolation.

5.2.4 Cell Projection of Stitch Cells

Rendering rectilinear, pyramid, and triangle prism cells which are produced by grid stitching, see Chapter 4, requires modifications to the standard cell projection approach. For these cell types, interpolated values cannot be obtained by linear interpolation on boundary faces followed by linear interpolation along ray segments.

To define interpolation for these cell types the *standard elements* shown in Figure 5.8 are considered. Interpolated values are computed by mapping all cells generated in the stitching process to the appropriate standard element. All cells generated in the stitching step have either triangular or quadrilateral faces. On boundary faces it must be ensured that interpolation yields consistent results, regardless of element type. Bilinear interpolation is used for quadrilateral faces and linear interpolation (based on the barycentric coordinates of the interpolated point) for triangular boundary faces.

Values in the unit cube, see Figure 5.8(a), are interpolated using standard trilinear interpolation which corresponds to bilinear interpolation when restricted to the boundary faces.

In the case of a triangular prism cell, see Figure 5.8(b), values at the vertices of the triangle containing the point $\mathbf{p}$ (*i.e.*, the triangle that is obtained by intersecting the prism with a plane being parallel to its two triangular end faces and containing $\mathbf{p}$) are computed by linear interpo-

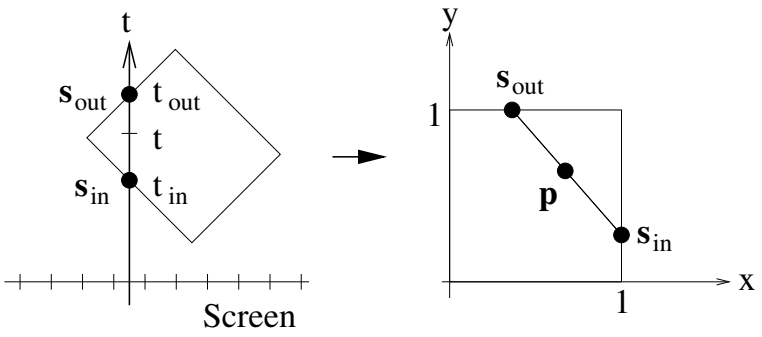


Figure 5.9: The transformation between a stitch cell and its standard element is linear. Line ray segments in the stitch cell are mapped to line segments in the standard element.

lation in x-direction. Subsequently, the value at $\mathbf{p}$ is obtained by linear interpolation within this triangle.

Interpolation in pyramid cells, see Figure 5.8(c), is done by a combination of bilinear and linear interpolation. To obtain a function value for a point $\mathbf{p}$, values along the four side edges emanating from the point $(0, 1, 0)$ are computed by linear interpolation: this step interpolates the values at the vertices of a square in the $y =$ constant plane containing $\mathbf{p}$. Subsequently, the value for $\mathbf{p}$ within that plane is obtained by bilinear interpolation in the $y =$ constant plane.

For tetrahedral cells, linear interpolation is used. The point $\mathbf{p}$ is expressed in term of its barycentric coordinates and these are used as interpolation weights. Linear tetrahedra are handled correctly by the standard cell-projection approach. In order to handle all cell types uniformly, tetrahedra are also handled by mapping them to the standard tetrahedron element.

To create ray segments, interpolated values must be computed along these segments. This is done by mapping each cell to its corresponding standard/unit element and using the interpolation function for that element. Mapping a cell to its associated standard element can be easily integrated into cell projection. For each vertex, its physical coordinates and corresponding coordinates in its standard element are stored. When cell faces are scan converted using physical coordinates, the standard-element coordinates are linearly interpolated on the face and stored along with depth values. Thus, points $\mathbf{s}_{\text{in}}$ and $\mathbf{s}_{\text{out}}$ are known for a ray segment in standard-element space along with its entry and exit parameter values t_{in} and t_{out}. All cells generated by my stitching approach, and the specific AMR meshes that I am dealing with, can be mapped to standard elements by a linear mapping: Line segments within a cell are mapped to line segments within a standard element. Thus, it is possible to obtain values along a ray segment by linearly interpolating the position in the standard element between $\mathbf{s}_{\text{in}}$ and $\mathbf{s}_{\text{out}}$ and using the standard-element interpolation function for this position, see Figure 5.9.

5.2.5 Progressive Cell Projection of AMR Data Using Stitch Cells

For constant and PLM interpolation the original grid is used to calculate interpolated values. Thus, for each cell, there exits a unique finer grid that refines it. Each cell of the used dual grid is defined by eight vertices. Each of these vertices corresponds to a cell of the original AMR

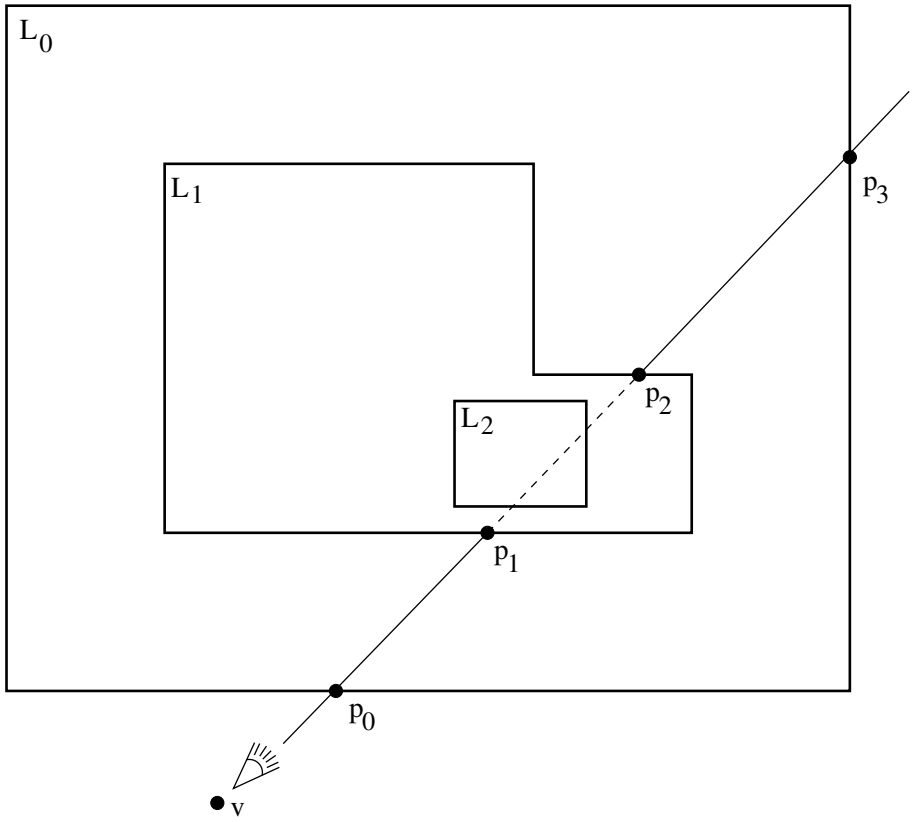


Figure 5.10: Progressive rendering of AMR hierarchy.

grid and can be refined by a different grid. It is no longer possible to specify a single grid that refines a given cell. However, it is still possible to specify, for each cell, whether it is refined by finer levels. If at least one of the original AMR cells corresponding to one of the vertices of a cell is refined, then that cell must be skipped. Unfortunately, this fact prevents refinement on the basis of single AMR grids. In a new approach, complete levels must be used when generating a refined image.

The top-down approach is modified in a similar way. The supplemental information added to each ray segment specifies the level in which a segment was created and the level that affects it (either the next finer level or no level) rather than specifying grid indices. Ray segments are merged only when they are adjacent, were created in the same level and are affected by the same level. Figure 5.10 shows a ray traced from a specific viewpoint. After rendering the root level L_0, the ray-segment queue corresponding to this ray contains three ray segments: one spanning from the entry point in p_0 to the beginning of the first level (p_1), one that is contained within the first level (from p_1 to p_2) and one after the exit point from the first level (from p_2 to p_3). The region from v to p_0 is "empty" and contains no cells. No ray segments are created for this region.

Using an approach of partitioning rays into segments that are affected by finer grids and those that are not, it is straightforward to refine an already rendered image by rendering a finer level. Before a finer level is rendered, all ray segments affected by the finer level are erased from the

ray-segment queues. When the level grid is rendered, the gaps in the ray segments resulting from this step are filled with more accurate ray segments, resulting in an overall improved image.

5.2.6 Level-dependent Transfer Functions

When rendering a hierarchical data set, it is often desirable to emphasize or de-emphasize certain levels. For example, it is possible that a coarser level is used to specify only the "context" within which a finer level resides, but otherwise this coarser level might be of little interest. In this case, the coarse level should not hide relevant information present in the finer level. One way to achieve this is to de-emphasize the coarse level and render it with lower opacity, *i.e.*, to scale the opacity portion of the transfer function by a level-specific constant. By specifying a constant for each level, and modifying the transfer function for that level accordingly, it is possible to specify how much a level influences the final image.

Another possibility to de-emphasize a level is to scale its color saturation. This can be done by converting RGB color values from the transfer function into HLS or HSV color space and scale the saturation by a second, level-dependent color map. Decreasing the saturation of a level does not prevent it from hiding details of a finer level, but this method adds the possibility to distinguish levels and illustrate their presence in a final image.

5.2.7 Results

Figure 5.11 shows a volume rendered image obtained by using cell-projection interpolating using the PLM. Data set, view point and transfer function are identical to the ones used during hardware-accelerated preview rendering, see Figure 5.5.

Figures 5.12 and 5.13 show results from rendering the "Argon Bubble" data set, which is courtesy of Center for Computational Sciences and Engineering (CCSE), Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, California. This data set is a time sequence resulting from simulating a shock wave passing through an Argon-bubble surrounded by another gas. The visualized scalar field is gas density. The simulation result is stored in AMR format with a $80 \times 32 \times 32$ root-grid resolution. In the initial time step, this grid is refined by 204 grids in a three-level hierarchy. Rendering all levels of the initial time step takes about two minutes and 23 seconds. Figures 5.12(b), 5.12(c) and 5.12(d) show the results from rendering one time step near the end of the simulation. This time step consists of three hierarchy levels containing 682 grids in total. Rendering the root level required approximately 57 seconds, rendering the first level one minute and 42 seconds and rendering the second level three minutes and 41 seconds. The improvement in image quality by using the finer levels is clearly visible.

Figure 5.13 shows another time step form the same data set. Rendering time was 40 seconds for the root level, one minute and 32 seconds for the first level and two minutes and 48 seconds for the second level. This time step consisted of 525 grids in total. Rendering times were one minute and 23 seconds for the root level, one minute and 55 seconds for the first level and four minutes for the second level. The quality improvement from using finer representations is clearly visible. (All time measurements were done on a 700 MHz Pentium III processor and using a Linux system.)

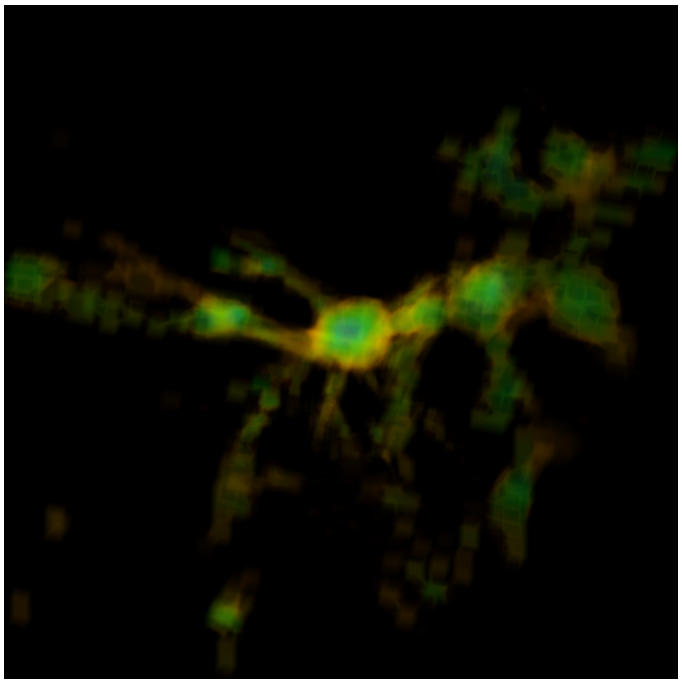
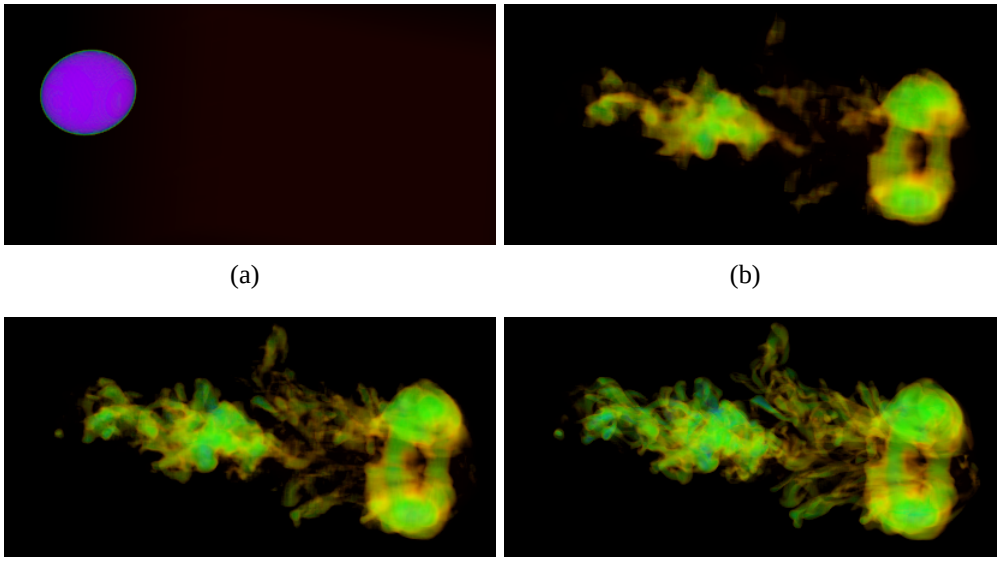


Figure 5.11: Final higher-quality rendering using cell-projection with interpolation via the PLM and with the same viewpoint and transfer function used for Figure 5.5.

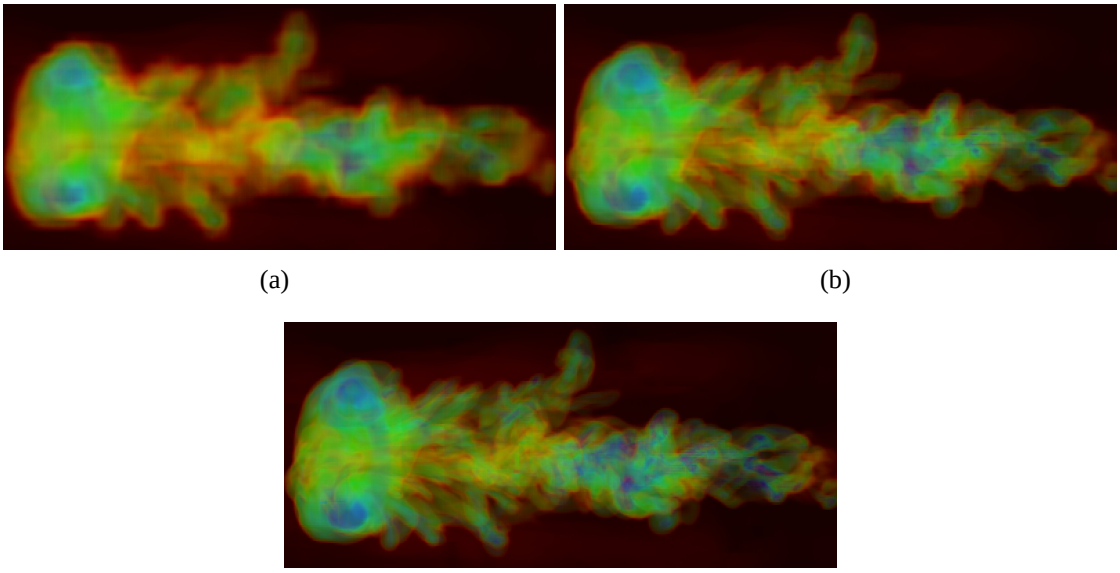
Figure 5.14 shows images generated from an astrophysical simulation of a star cluster. Figures 5.14(a), 5.14(b) and 5.14(c) show images resulting from rendering one, two and three levels. Here, the quality improvement is not so obvious, because features of the coarse root level hide details from the finer levels. In Figure 5.14(c), the opacity of the root level is scaled by a factor of 0.6 and the opacity of the first level by a factor of 0.8. Details of the finer level are clearly visible while retaining features from the coarse levels as orientation aid. In Figure 5.14(d), the saturations of root and first level are scaled by a factor of 0.2 in addition to the opacity weights. This further de-emphasizes these levels and allows me to clearly distinguish between the details in the third level and the "context" provided by the first two levels.



(a) (b)

(c) (d)

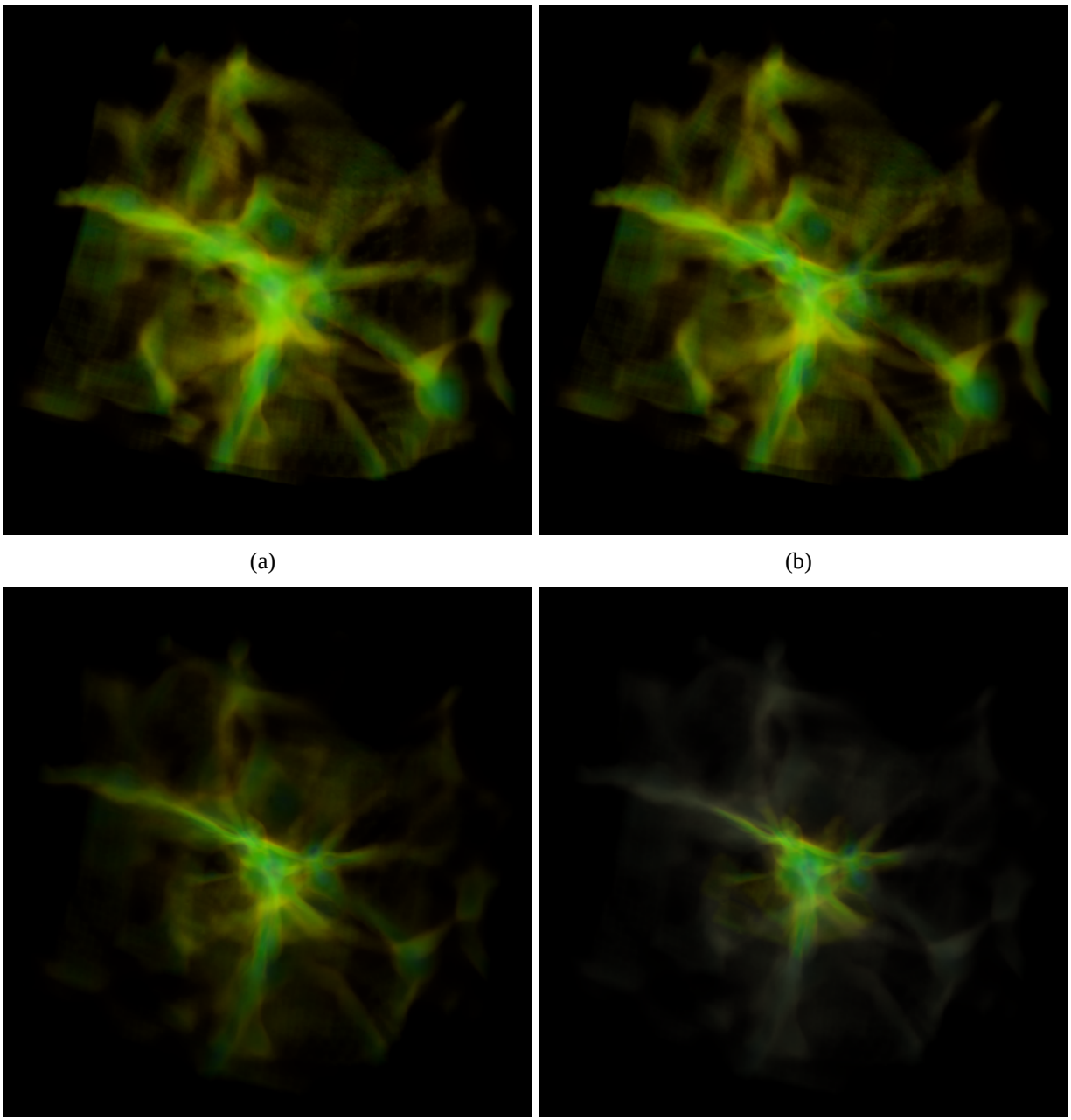
Figure 5.12: Images generated from “Bubble” data set. (Data set courtesy of Center for Computational Sciences and Engineering (CCSE), Ernest Orlando Lawrence Berkeley National Lab, Berkeley, California) (a) Initial time step at full resolution. (b) Time step near end of simulation using only coarsest level of hierarchy. (c) Time step near end of simulation using two levels of hierarchy. (d) Time step near end of simulation using all levels of hierarchy.



(a) (b)

(c)

Figure 5.13: Different view of the “bubble” data set. (Data set courtesy of Center for Computational Sciences and Engineering (CCSE), Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, California) (a) Coarsest level only (b) Two levels (c) Entire AMR hierarchy



(a) (b)

(c) (d)

Figure 5.14: Images generated from cosmology simulation. (Data set courtesy of Greg Bryan, Massachusetts Institute of Technology, Theoretical Cosmology Group, Cambridge, Massachusetts) (a) Coarsest level only. (b) Three levels. (c) Three levels using opacity weights 0.6, 0.8 and 1 for levels 0, 1 and 2, respectively. (d) Three levels using same opacity weights as in Figure 5.14(c) and a saturation weight of value 0.2 for levels 0 and 1.

5.3 Parallel Volume Rendering of Adaptive Mesh Refinement Data

5.3.1 Introduction

Based on an optimized version of software cell-projection presented in Section 5.2.2 I have developed a framework for parallel volume rendering of AMR data was developed. An AMR hierarchy is partitioned using a k-d tree [5]. This partition is view-independent and computed offline in a preprocessing step. Several partition strategies have been developed and compared, which are briefly summarized.

Uniform root-level subdivision ignores the hierarchical nature of AMR data and partitions a root level into blocks of constant size. Refined cells are handled during rendering by recursive descending into finer levels.

Weighted root-level subdivision partitions a root level into blocks at approximately constant computational cost. The AMR hierarchy is only considered to compute weights. Locations for subdivision are chosen independently from boundaries of refining grids. During rendering refining grids are handled by descending recursively.

Homogeneous subdivision subdivides AMR levels recursively until each part only covers one grid of a given level, *i.e.*, until it corresponds to a region represented at constant resolution. The resulting grid parts are distributed evenly among processors.

Weighted homogeneous subdivision partitions AMR levels in the same way as homogeneous subdivision. The computational cost for rendering a constant-resolution region is estimated and associated with that region as its *weight*. Grid parts are distributed among processor such that the sum of associated weights is approximately the same for all processors.

Images are computed on parallel supercomputers or PC clusters. This class of machines is the same as that used to perform the AMR simulations. Thus, the used resources are readily available to users. The framework supports rapid development and testing of new distribution strategies and volume rendering techniques.

5.3.2 Design Considerations

The foremost design concern was efficiency and rendering times. The particle light model was chosen as it leads to efficient implementations. Within cells, constant interpolation, *i.e.*, the sample value located at the cell center is assigned to all positions within the cell, is used. This allows an exact evaluation of the light-model and preserves the AMR hierarchy in rendered images. To achieve more efficiency, orthographic projection is chosen over perspective projection. Based on my experiences with the software-based cell-projection approach described in Section 5.2.2, an optimized renderer was developed. Instead of sorting ray segments in priority queues, cells are rendered in back-to-front order. Newly generated ray segments are always adjacent to already computed ray segments and can be composited directly in the frame buffer eliminating the

need to sort ray segments. Another advantage of this method is that it allows to avoid duplicate scan conversion of a cell's boundary faces. When rendering unsorted cells, back-facing and front-facing faces must be rendered to determine correct ray-segment length. In contrast, when rendering presorted cells, it is sufficient to render the front-facing faces of a cell. All back-facing faces are already rendered as front-facing faces of cells "behind" the current cell.

Parallelizing volume rendering can be done in image space or in object space, see Crockett [16]. Image-space parallelization subdivides the image plane to distribute computing among multiple processors. Each processor renders a subset of pixels in an image. Object-space parallelization subdivides the domain of a data set and assigns grid cells to processors. Object-space parallelization was chosen, as the hierarchical nature of AMR data facilitates efficient subdivision of the grids. Cell-projection was chosen as an object-space rendering method, as it leads to an elegant implementation of subdivision of the domain. Furthermore, using cell-projection eases reaction to changes in resolution, *i.e.*, it is possible to render finer grids at a higher resolution.

For implementation of the parallel renderer the Message Passing Interface (MPI) library was chosen over the Parallel Virtual Machine (PVM) framework. MPI is commonly used in AMR simulations, thus making my framework more compatible with other applications, including numerical simulation. Furthermore, MPI is a de facto standard for parallel supercomputers. Vendor-specific adaptations for different architectures exist, supporting the utilization of specific hardware optimizations by linking to a vendor-provided library. Instead of adopting the classic master-slave model, a symmetric implementation was chosen to avoid communication bottlenecks. Each processor computes the complete distribution of grid parts and selects a subset based on its index. However, a binary-tree image compositing scheme is used that pairs processors in each compositing step. In each step, one processor of each pair receives an intermediate partial image from its "neighbor" and performs a compositing operation. The final composited image resides in the buffer of processor zero.

Cell Sorting and Front-face Determination

Correct back-to-front order is determined based on view direction using the same criteria as in the hardware-based approach described in Section 5.1.2. Cells are enumerated by three nested loops, one loop for each axis. The order according to which axes are handled is arbitrary. Along each axis, cells must be handled in correct order. For each loop, this order can be determined based on the sign of the component of the vector $\mathbf{tv}$ (a vector directed toward the viewer), according to the axis handled by the loop. If it is positive, cells are enumerated in ascending axis direction. If it is negative, cells are enumerated in descending axis direction. If if is zero, an arbitrary choice is made.

Before rendering cells and generating ray segments, all cell faces lying on the up to three back-facing boundary-faces of the overall AMR grid that are not view-perpendicular must be scan-converted. These are the back-facing faces of cells that do not lie in front of any grid cell. Figures 5.15 (a)–(c) illustrate the procedure for a choice of $\mathbf{tv}$ where all components are positive. If one component of $\mathbf{tv}$ is zero, the corresponding face is perpendicular to the viewing direction and discarded. Subsequently, the front facing faces of all cells are scan-converted. Ray segments are generated and composited in the frame buffer. Figure 5.15 shows the order of scan-conversion

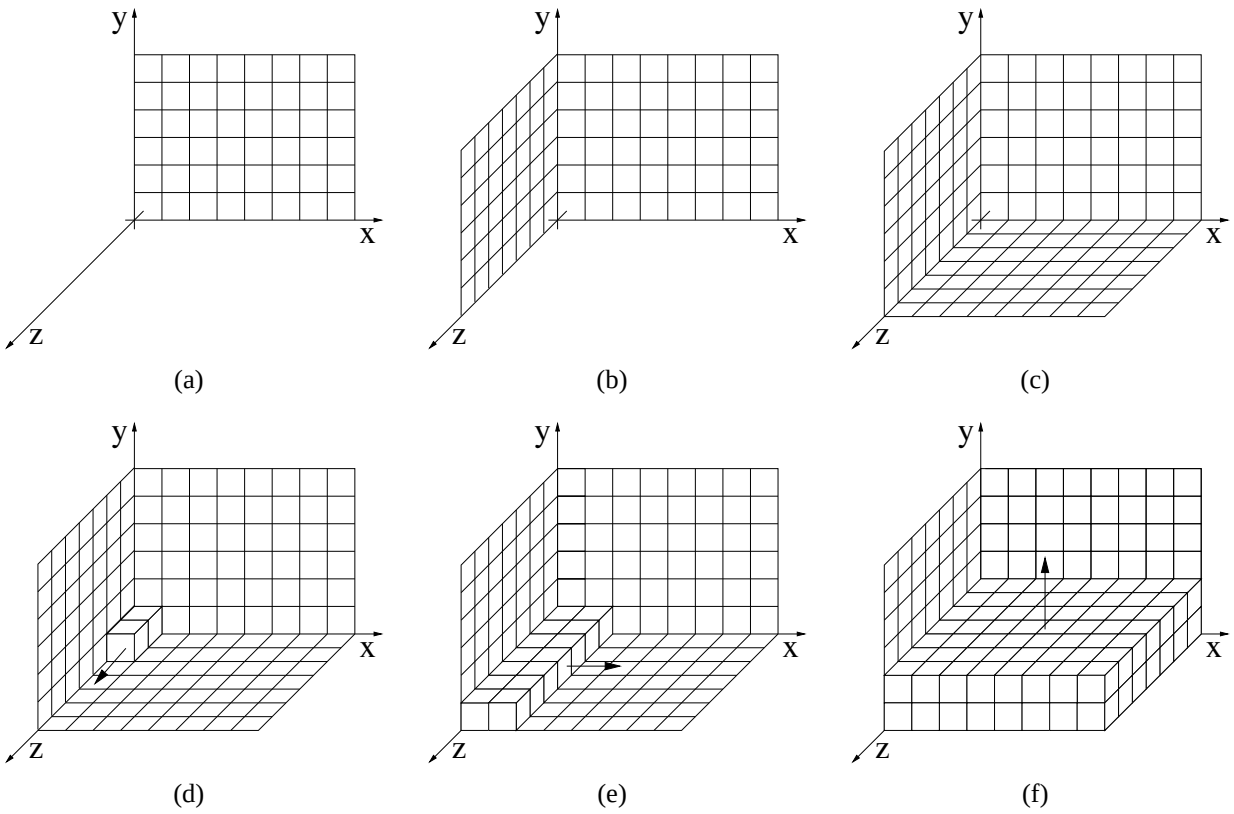


Figure 5.15: Rendering order of grid cells — all components of $\mathbf{tv}$ being positive. First, all back-facing faces of the first layer of cells in each direction are rendered (a) – (c). Second, all cells are rendered. The order in which axes are handled (first-$z-$-then-$x-$-then-y order) is arbitrary. Only the order according to which cells are handled along an axis is important.

used for boundary faces and cells when all components of $\mathbf{tv}$ are positive.

Boundary Face Scan-conversion

Cell boundary faces are rendered using a modified version of the polygon scan-conversion algorithm developed by Gordon *et al.* [29] that is based on a method developed by Kaufman [43]. Before rendering a polygon, its vertices are projected onto the viewing plane, and point coordinates are rounded to integers. During the scan-conversion process it is assumed that coordinates are specified counter-clockwise. Gordon *et al.'s* method starts by determining "critical points" of a polygon, *i.e.*, vertices that constitute a local minimum or are first of a set of vertices that together form a local minimum in y-direction. Boundary faces of rectilinear cells are convex quadrilaterals and have only one minimum. If two adjacent vertices share the same value for the y-coordinate, *i.e.*, if they are connected by a horizontal line segment, the vertex with the lower index is considered to be the critical point. During the determination of critical points, polygons that span one pixel in y-direction are detected and discarded.

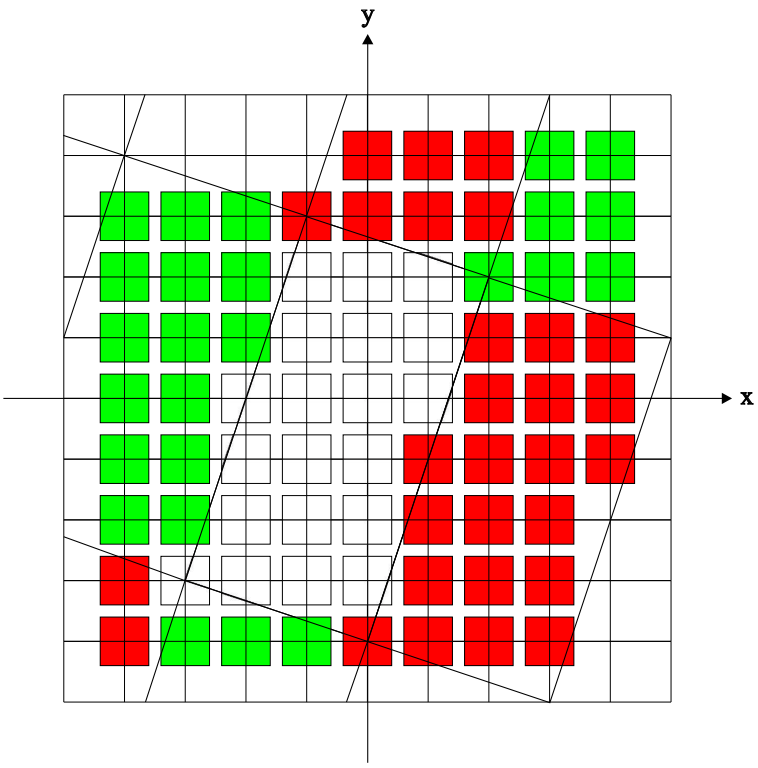


Figure 5.16: Scan-converted polygon illustrating rules used to determine whether a pixel belongs to the polygon.

The algorithm starts by inserting the left minimum and right maximum edge originating from the critical point vertex into an active-edge table (AET). During the scan-conversion process this data structure holds the left and right edge intersecting the current scan-line. For general polygons, considered by Gordon *et al.* [29] this is an array, as the scan-line can intersect several polygons. The scan-converter is optimized for convex quadrilaterals and only stores two pointers to AET elements since a convex quadrilateral intersects a scan-line only twice, except for horizontal boundary lines coinciding with a scan-line. For each scan-line, x-coordinates and depth on the left and the right side of the polygon are calculated by linear interpolation. Depth information is not rounded, as exact values are needed for the determination of ray-segment lengths. If a scan-line consists of only one pixel of the polygon it is discarded; otherwise, depth values are computed for all pixels between the x-coordinates by linear interpolation. Ray segments are created by reading the previous depth value and applying the illumination model. If a scan line coincides with the end of an edge, the corresponding pointer referring to the AET is replaced with its successor until that turns down.

When generating images with a cell-projection method it is important that rasterized polygons sharing an edge do not overlap. Thus, special care must be taken at polygon boundaries. The following rules are used:

R1) Integer intersection points of a polygon edge with a scan-line belong to a polygon, if they lie on its left edge. If they lie on its right edge, they do not belong to the polygon.

R2) Non-integer intersection points of a polygon edge with a scan-line are rounded down. The corresponding pixel belongs to a polygon if it lies on its right edge. If it lies on its left edge, it does not belong to the polygon.

R3) If a pixel corresponds to intersection points on the left and right edges of a polygon it lies outside the polygon.

The white center polygon in Figure 5.16 illustrates these rules: The pixel at the lower-left corner of the polygon has an integer intersection point and lies on its left edge. Consequently, according to R1, it belongs to the polygon. According to R1, the pixel at the upper right corner of the polygon does not belong to the polygon. Considering R2, all pixels with non-integer intersection points on the left and lower polygon edge do not belong to the polygon. (They belong to the neighboring polygon.) All pixels bordered by the upper and right polygon edges do belong to the polygon. The pixel at the upper-left corner lies on the left and the right edges of the polygon and does not belong to the polygon (R3). During scan-conversion, a list of all positions that are modified, *i.e.*, covered by a cell, is maintained . This list is used in the compositing scheme, see Section 5.3.4, and to speed up clearing the frame buffer by only erasing pixels modified during rendering.

Ray-segment Generation

Constant interpolation is used within individual cells, *i.e.*, the sample value associated with a cell is assigned to all positions in the cell. Consequently, all points in a cell have the same optical properties, *i.e.*, emission color and opacity. It is possible to solve the differential equations for light absorption and emission analytically in a cell and obtain "correct" opacity and emission values for a ray segment intersecting the cell. Each ray segment in a cell is characterized by an entry parameter value t_{in}, *i.e.*, the distance from the viewing plane at which a ray "enters" the cell measured along the ray, and an exit parameter value t_{out}, *i.e.*, the distance from the viewing plane at which the ray "exits" the cell. The value of t_{in} is obtained by scan-converting the front-facing faces of a cell. The value of t_{out} is read from the frame buffer containing the results from scan-converting the front faces of cells (behind the current cell) that coincide with the back-facing faces of the current cell. Emission color and opacity are defined by the cell's associated scalar value via a transfer function.

5.3.3 Partitioning and Load-balancing

Overview

A domain partition is stored as a k-d tree [5]. A k-d tree is a generalization of a binary search-tree to arbitrary dimensions. Each level partitions a domain in two regions along an axis-perpendicular plane. The "left" sub tree corresponds to points in space whose coordinates in

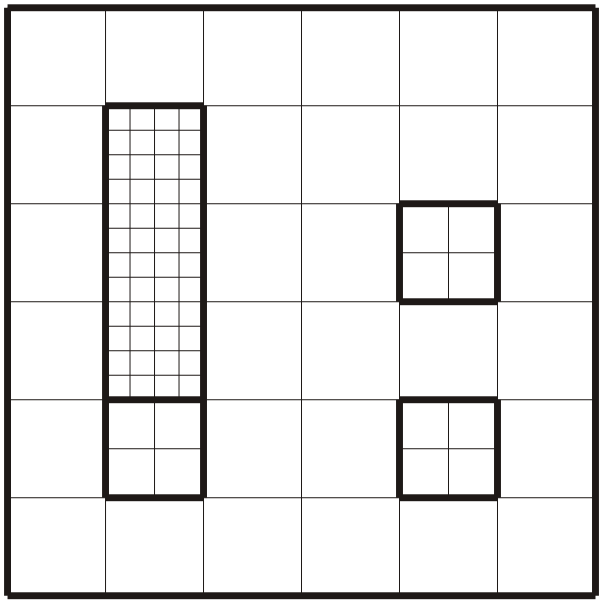


Figure 5.17: Example grid used to illustrate subdivision strategies.

partition direction have values smaller than or equal to the partition position. The "right" sub tree corresponds to points whose coordinates in partition direction have values larger than the partition position. The subdivision direction is usually alternated between the three coordinate axes in the 3-d case. I skip subdivision directions, when no "sensible" subdivision position along that direction exists. When using an object-space-based subdivision for parallelizing volume rendering, regions must be rendered in correct order. Using k-d trees makes it possible to determine this order simply. At each node of the k-d tree, the domain is subdivided along an axis-perpendicular plane. The compositing order can be determined by considering the component of $\mathbf{tv}$ corresponding to the partition direction. If it is positive the left sub tree must be rendered first; if it is negative the right sub tree must be rendered first; and if it is zero both sub trees can be rendered in arbitrary order. I assume that subdivision schemes are view-independent. It is sufficient to compute a k-d tree subdivision once per data set. Subdivisions are computed offline in a pre-processing step. To assign regions of the domain, *i.e.*, leaves of the k-d tree, to individual processors they are numbered in rendering order. A set of sequentially adjacent regions is assigned to each processor.

Uniform Root-level Subdivision

Given an AMR hierarchy, this scheme constructs a k-d tree with a user-specified number of levels. Each node of the tree splits its associated region into two parts of nearly equal size, *i.e.*, number of cells. Figure 5.18 shows uniform subdivision of the AMR hierarchy from Figure 5.17.

Since uniform subdivision ignores grid boundaries, refined cells must be handled during rendering. This is done by recursively descending the hierarchy. While rendering a data set, a

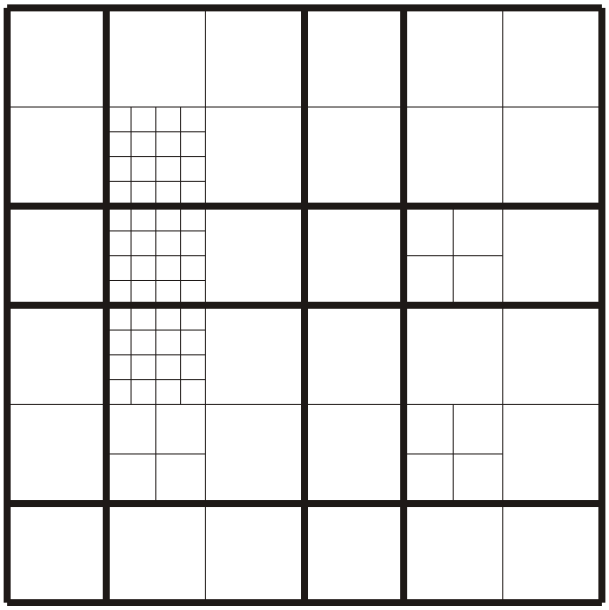


Figure 5.18: Uniform subdivision of root level into equal-sized blocks.

test is performed for each grid cell checking whether it is refined by the next finer level. If a finer level exists, the r^3 (r being the refinement ratio) refining grid cells are rendered instead of the current coarser cell. The correct rendering order of refining cell is determined using the criteria described in Section 5.1.2. Each refining cell is checked recursively to determine potential further refinement.

Weighted Root-level Subdivision

Similarly to uniform subdivision, this scheme ignores grid boundaries of an AMR hierarchy during subdivision. The goal of this approach is to obtain a subdivision of a given AMR hierarchy into regions that will imply approximately equal computational cost. Each region is associated with an estimate of computational cost for rendering, used as a weight. A subdivision plane is chosen using a greedy method as follows: Initially, the subdivision plane is placed in the middle of the current domain. Weights are computed for the two subdomains. If both subdomains have equal weight, the subdivision plane has optimal position and the algorithm terminates. Otherwise, the plane is moved into the subdomain with the larger associated weight. Moving the plane in this way decreases computational cost for that subdomain while increasing computational cost for the other one. The weight difference is calculated before and after moving the plane, and the plane is moved as long as it decreases the weight difference. The algorithm terminates when moving the plane increases the difference instead of decreasing it, or the partition plane would reach the border of a subdomain.

The computational cost of a subdomain is estimated based on the number of cells in it. It is also necessary to consider the fact that rendering refined cells is computationally more expensive

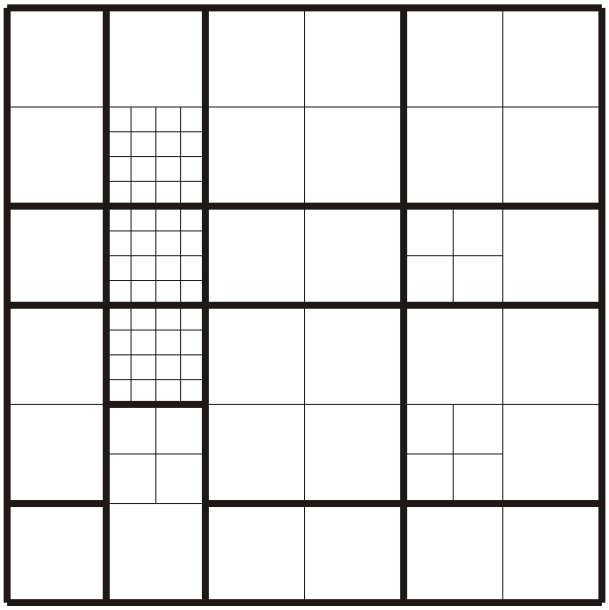


Figure 5.19: Weighted subdivision of root level, ignoring grid boundaries.

CPU type	c_0	c_1	c_2
1.0 GHz AMD Athlon	1.00	0.60	0.50
1.2 Ghz AMD Athlon	1.00	0.65	0.54
1.4 GHz AMD Athlon	1.00	0.71	0.58
2.4 Ghz Intel Xeon	1.00	0.63	0.54
375 MHz IBM Power 3	1	0.77	0.71

Table 5.2: Constants for weighted distribution for different processors.

than rendering unrefined cells. Therefore, a weight of one is assigned to unrefined cells of the root level, *i.e.*, $c_0 = 1$. Based on an application specific benchmark it is possible to determine relative weights for refined cells. Table 5.2 shows weights for an AMR hierarchy consisting of three levels. The constants specify the times necessary to render a single cell of a given AMR level with respect to rendering times for a single cell of the root level. These constants are measured by rendering a cell of the appropriate level from a viewing direction of $t_v = (1, 1, 1)$. Viewing a cell in this direction no cell faces are axis-perpendicular. A maximum number of faces must be rendered and the "footprint" of the cell on the screen has maximum size. The associated weight w of a subdomain is

$$w = \sum_{l=0}^{\#\text{Level}} n_l c_l \ , \tag{5.6}$$

where n_l is the number of level l cells. This sum is computed by recursively descending in the

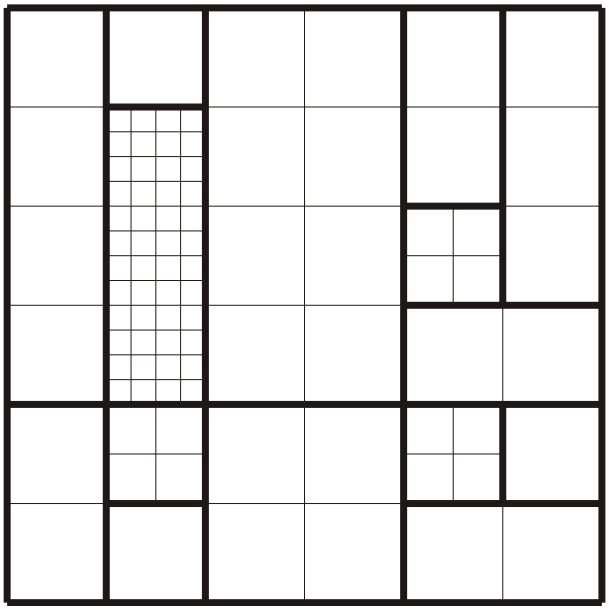


Figure 5.20: Homogeneous subdivision of AMR hierarchy.

hierarchy. Figure 5.19 shows weighted subdivision of the root level for the AMR hierarchy from Figure 5.17, using relative cell weights of $c_0 = 1$, $c_1 = 0.75$ and $c_2 = 0.7$.

Homogeneous Subdivision

Both sub-division strategies discussed so far ignore the hierarchical nature of AMR data during subdivision. Only the impact on computational cost for rendering a grid part is considered when using weighted root-level subdivision. Resulting regions usually encompass several grids of the original hierarchy, leading to data duplication and poor memory utilization. By considering grid boundaries during the subdivision step, it is possible to partition an AMR hierarchy into "homogeneous" blocks, *i.e.*, blocks represented at constant resolution. Each block contains only cells from one grid of the original AMR hierarchy. This property allows me to avoid data duplication. Due to the homogeneous nature of blocks, it is possible to render them efficiently, avoiding tests for refinement of individual cells and recursion.

Subdivision of an AMR data set uses only information about the hierarchical structure of AMR data. Actual data values for individual grids do not need to be loaded, and subdivision can be performed on a single machine, requiring only a small amount of memory. The k-d tree is constructed level by level. For level l, all leaf nodes of the current k-d tree are located that correspond to grids of level $l-1$. Each of these leaves is replaced by a k-d tree that is constructed as follows: All grids of level l that overlap the leaf region, *i.e.*, the region associated with the current leaf, are determined. Since grids may only partially overlap the leaf region, they are clipped against the leaf region to obtain the grid part contained in the leaf region. Along the current subdivision direction, every position in subdivision direction is stored where a refining

grid starts or ends.

After sorting the resulting list and removing duplicate elements, the middle element of the resulting list is chosen as subdivision position. (If the list contains an even number of elements, the smaller of the two middle elements is chosen as subdivision position. If the list is empty, the corresponding subdivision is skipped direction.) For each of the two regions associated with a subtree of the current leaf, all grids are found that overlap that region and cliped against the boundaries of that region. Alternating between the three axis directions, this process is repeated recursively until all leaf regions are homogeneous and overlap only a single grid. For the root level, construction starts with an empty tree that covers the complete domain and construct a k-d tree analogously to creating the tree for a leaf.

By terminating k-d tree construction after a user-specified fixed level-number it is possible to render only a part of an AMR hierarchy. Figure 5.20 shows the results of homogeneous subdivision of the AMR hierarchy from Figure 5.17. Grid parts are numbered in back-to-front order and distributed among processors. Each processor loads the complete partition information and renders nearly the same number of sequentially numbered grid parts.

Weighted Homogeneous Subdivision

Weighted homogeneous subdivision uses the same k-d tree subdivision as homogeneous subdivision. Instead of distributing resulting grid parts evenly among processors, an estimate of computational rendering cost is used as a weight for each leaf of the k-d tree. This weight is obtained by multiplying the number of grid cells in the leaf region by the weight of a single cell of the appropriate level. The weight of a single cell is the same as the one used in weighted root-level subdivision, see Section 5.3.3. Regions are distributed among processors using a greedy method. To achieve nearly equal processor utilization, each processor needs to render regions with a total weight of $w_{\text{Ideal}} = \frac{w_{\text{Total}}}{n_{\text{Processors}}}$, where w_{Total} is the computational cost for rendering the complete AMR hierarchy, *i.e.*, the sum of all weights of all leaves of the k-d tree. Each processor has an assigned set of sequentially adjacent leaves. Processor p selects its assigned regions as follows: If k is the last leaf rendered by processor $p-1$, processor p adds the remainder of that region, *i.e.*, the part that was not rendered by the previous processor, to its "assignment list." (An exception to this rule applies to the first processor. It does not need to render any partial regions.) Starting with region $k+1$, processor p adds regions to its assignment list until the weight of the current region exceeds the difference $w_{\text{Ideal}} - w_{\text{Curr}}$. During each step, w_{Curr} denotes the sum of weights of all regions already assigned to processor p.

To achieve a more uniform distribution of weights, this region is subdivided as follows: First, the direction perpendicular to the plane consisting of the least number of cells is chosen as partition direction. The difference $w_{\text{Ideal}} - w_{\text{Curr}}$ is divided by the weight of a slice in partition direction (*i.e.*, , the number of cells in the slice multiplied by the cell weight of the appropriate level). The result is rounded to obtain the number of slices rendered by the current processor. The remaining slices are rendered by the next processor. (An exception to this rule applies to the last processor which renders all remaining regions.)

Each processor computes assignments for all processors. This avoids the need for waiting for the previous processor to finish its own assignment computation. The index k of the last region

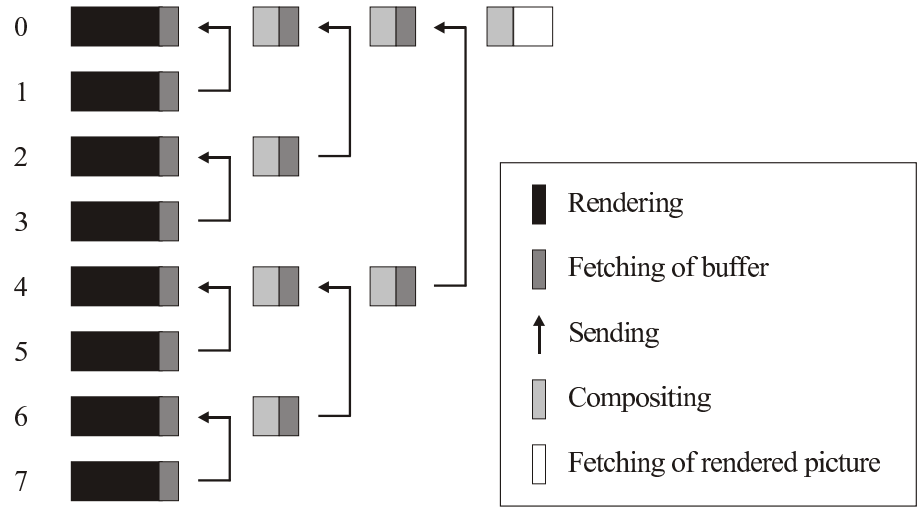


Figure 5.21: Parallel compositing scheme.

rendered by the previous processor and a potential remainder of a subdivided region are determined locally. Performing this computation in parallel, avoids added time for communication between processors.

5.3.4 Compositing

When all regions are rendered, the partial images are composited. Compositing is done by using alpha blending/compositing of the partial images, see Porter and Duff [72]. The compositing scheme is illustrated in Figure 5.21. In the first step, each odd processor sends its partial image to its lower-indexed neighbor processor that performs the compositing operation. In each subsequent step i only those processors that composited an image in the previous step are considered, *i.e.*, processors with an index of the form $k2^i$. Each processor having an associated odd value of k sends its intermediate partial image to processor $(k-1)2^i$ which performs the next compositing operation. Because regions are assigned to processors in back-to-front order, each processor can composite the partial image received from the other processor "over" the region in its own buffer. At the end of the compositing process, the final image is resides in the buffer of processor zero. When transferring partial images between processors for compositing, only pixels that have been altered during rendering are transmitted: position, color and alpha value for each altered pixel are transferred. In the "fetch buffer" stage this representation is converted to a bitmap.

5.3.5 Results

The distribution strategies were tested by rendering the last time step of the "Argon-bubble" data set described in Section A.5.2.7 This time step is stored in AMR format using a hierarchy consisting of 885 grids in three levels. All grids in total consist of 1401504 grid cells. Homoge-

Subdivision Strategy	Time [s]	Speedup
Uniform	14.90	2.56
Weighted Root-level	14.67	2.61
Homogeneous	12.35	3.10
Weighted Homogeneous	11.95	3.20

Table 5.3: Rendering times on Linux Cluster using four CPUs.

neous subdivision of the AMR hierarchy yields 6002 grid regions. Figure 5.22(a) shows the grid structure. Figure 5.22(b) shows the final volume-rendered image.

Distribution strategies were tested on the following machines:

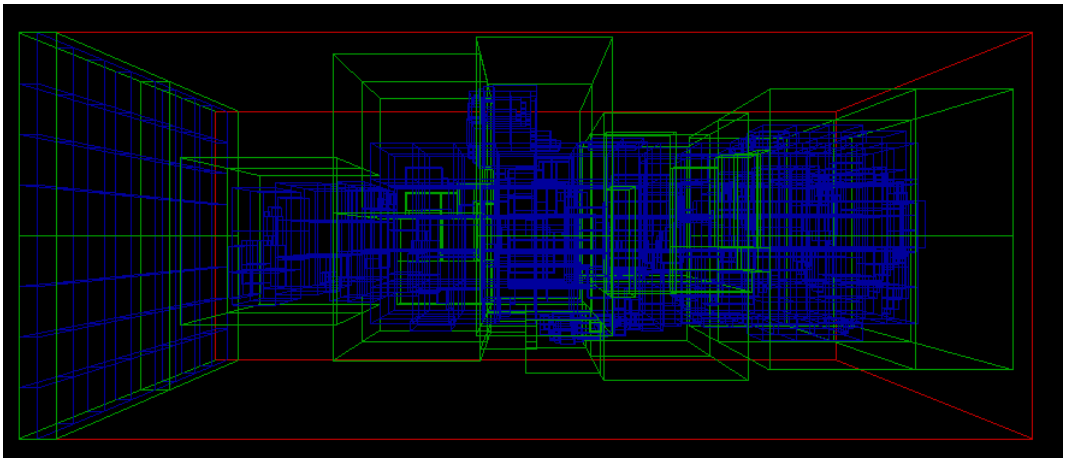
Linux Cluster This configuration is a Linux cluster consisting of four 1.2 GHz Dual-Athlon machines connected via a Gigabit network. For measurements with four processors, a single CPU was used on each machine. For measurements with eight processors, both CPUs on each machine were used. Each machine has 512 MB main memory.

Shared-memory machine This is a PC-based server equipped with two 2.4 GHz Intel Xeon CPUs using hyper-threading to obtain four "virtual" CPUs. The used machine has a total memory of 2 GByte RAM. A version of MPICH that supports the shared memory environment on that machine was used.

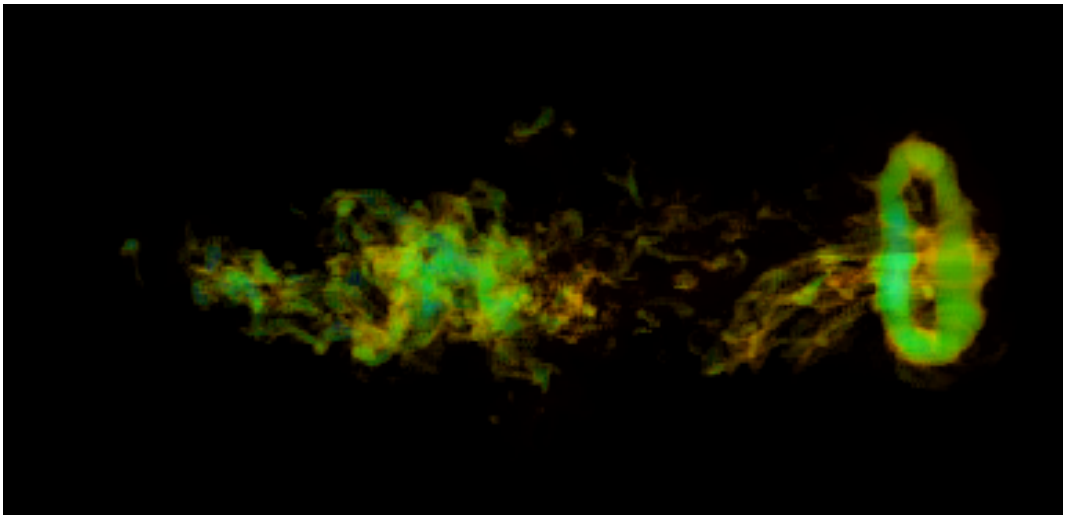
IBM SP2 Seaborg is a 10 Teraflop IBM SP RS/6000 located at NERSC's high-performance computing facility. It consists of 416 NightHawk II nodes. Each node contains 16 IBM Power3+ processors running at 375 MHz and 16–64 GBytes of shared memory. The nodes are interconnected using dual 150 Megabyte/s SP/"GX BusColony" switch adaptors forming a fat-tree topology. IBM's native MPI implementation was used.

Figures 5.23 – 5.26 show processor utilization for rendering on a four-processor Linux cluster. As expected, uniform subdivision achieves an uneven utilization of processors. Weighted root-level subdivision achieves a comparatively even processor utilization, but it requires longer rendering times than subdivisions working on homogeneous grid parts. This behavior is due to the overhead by recursively descending into the hierarchy. Recursive descend also causes non-local memory access pattern resulting in poor cache utilization. Working only on data of a single grid and avoiding overhead due to data inhomogeneity, homogeneous subdivision performs better than weighted root-level subdivision, even though computational cost is not as evenly distributed. Weighted homogeneous subdivision resolves this problem and achieves good rendering speed while near-uniformly utilizing all processors.

Tables 5.3 – 5.5 show rendering times and speedups for rendering on a Linux cluster and a shared memory machine. Speedups are measured with respect to rendering the data on a single processor using homogeneous subdivision. As expected, weighted homogeneous subdivision leads to best results of all considered subdivision schemes. It is important to note, that times



(a)



(b)

Figure 5.22: (a) Grid structure of "Argon Bubble." The hierarchy consists of 885 grids in three levels with a root grid of $80 \times 32 \times 32$ cells. (b) Volume-rendered image of "Argon Bubble." (Data set courtesy of Center for Computational Sciences and Engineering (CCSE), Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, California)

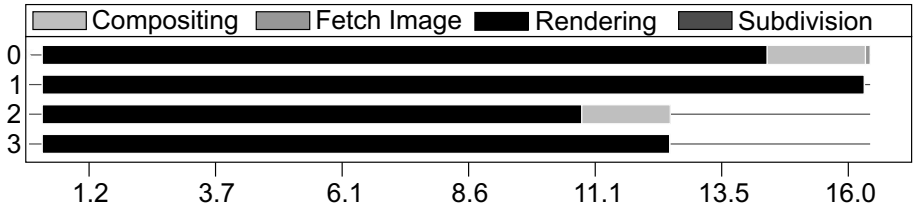


Figure 5.23: Processor utilization for uniform root-level subdivision.

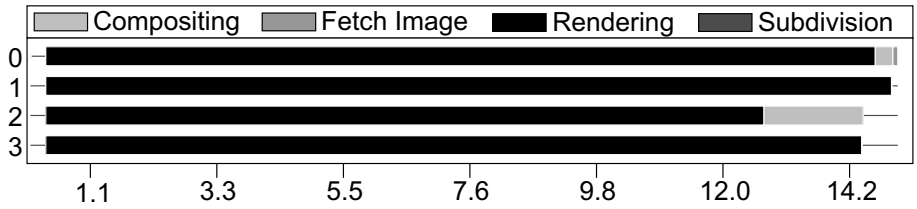


Figure 5.24: Processor utilization for weighted root-level subdivision.

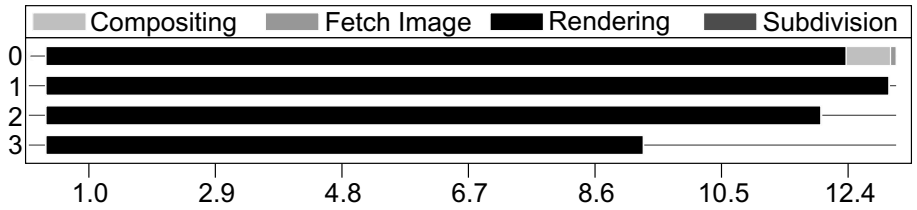


Figure 5.25: Processor utilization for homogeneous subdivision.

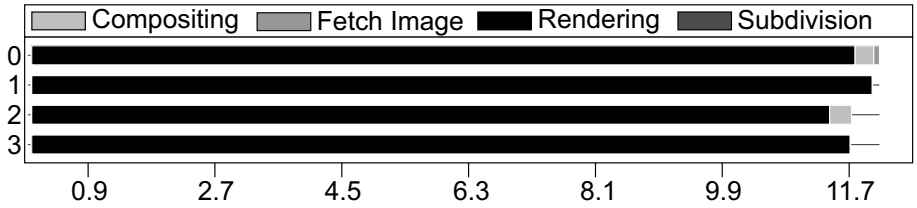


Figure 5.26: Processor utilization for weighted homogeneous subdivision.

Subdivision Strategy	Time [s]	Speedup
Uniform	7.83	4.89
Weighted Root-level	7.50	5.10
Homogeneous	7.10	5.39
Weighted Homogeneous	6.21	6.16

Table 5.4: Rendering times on Linux Cluster using eight CPUs.

Subdivision Strategy	Time [s]	Speedup
Uniform	14.76	2.59
Weighted Root-level	14.60	2.62
Homogeneous	12.16	3.14
Weighted Homogeneous	11.23	3.40

Table 5.5: Rendering times on shared-memory machine.

No. of CPUs	Weighted Root-level Time [s]	Speedup	Homogeneous Time [s]	Speedup	Weighted Homogeneous Time [s]	Speedup
1	142.10	1.00	125.45	1.00	124.98	1.00
4	39.87	3.56	36.25	3.46	33.83	3.69
8	21.67	6.55	19.81	6.33	17.89	6.98
16	11.61	12.23	12.93	9.70	8.79	14.21
32	7.88	18.03	7.31	17.16	6.24	20.02
64	5.42	26.21	6.22	20.16	2.97	42.08
128	3.09	45.98	3.83	32.75	1.98	63.12
256	2.07	68.64	1.48	84.76	1.53	81.68
512	1.66	85.60	1.27	98.77	1.37	91.22

Table 5.6: Rendering times on IBM SP2.

vary between subsequent runs of the framework, and timings are only accurate within approximately one second. Considering these facts, the speedup achieved by weighted homogeneous subdivision is satisfactory.

Table 5.6 shows rendering times on an IBM SP2. These measurements are of "strong scaling" behavior whereby the problem size remains fixed as the number of processors is increased. This typically results in less flattering scaling efficiency than if the problem size was scaled to be proportional to the number of processors as is the case for "weak scaling" studies. Timing granularity for large-scale parallel applications is typically on the order of approximately one second. Thus, results utilizing more than 128 processors on that system have a lower degree of confidence than the smaller tests. Starting with 256 processors, homogeneous subdivision surprisingly performs better than weighted homogeneous subdivision. The difference in the performance of the models for the very large scale runs is less than the timing granularity, so I only have limited confidence that these effects are actually real rather than being timing artifacts. However, that being said, I believe these timings are consistent with the observation that the time required for assigning regions to processors is higher for weighted homogeneous subdivision. While rendering time on each processor decreases for a larger number of processors, the time spend computing the subdivision becomes the dominant computational cost. It is also possible that the granularity of work that can be assigned becomes large compared to the total amount of work that is assigned to each processor — offering less benefit to these fine-grained optimizations.

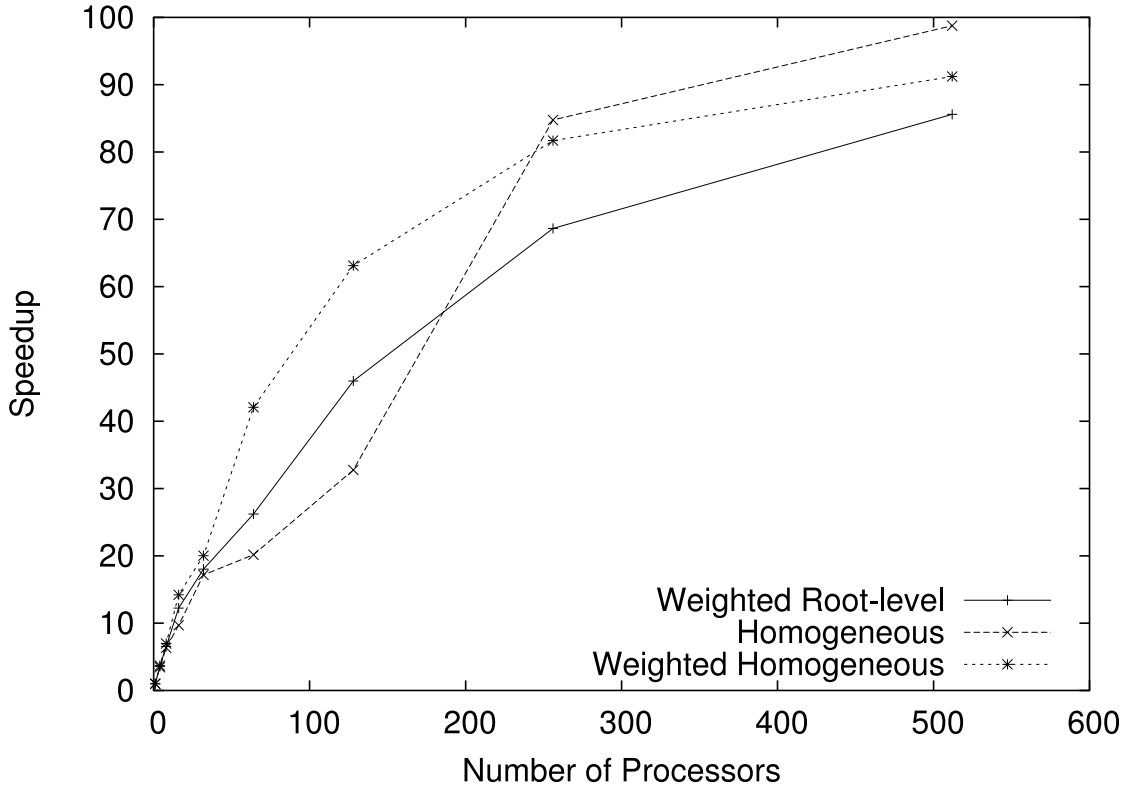


Figure 5.27: Speedup as function of number of processors (IBM SP2).

Chapter 6

Topology-based Scalar Data Analysis and Visualization

6.1 Introduction

Trivariate data is commonly visualized using isosurface extraction or direct volume rendering. When exploring scalar fields by isosurface extraction it is often difficult to choose isovalues that correspond to "interesting" isosurface behavior, *i.e.*, result in isosurfaces conveying "meaningful" information about the data. Features of a scalar data set can be easily missed when certain isovalues are not considered. The significance of visualizations using direct volume rendering depends on the choice of good transfer functions. Understanding and using isosurface topology can help in identifying "relevant" isovalues for visualization via isosurfaces and can be used to automatically generate transfer functions.

Isosurface topology provides insight into the fundamental structure of isosurface behavior across isovalues. By treating a volume data set as "height" function over a 3-d domain and using Morse theory, see Section 2.1.2, in four dimensions, it is possible to track topological changes of an isosurface by looking at critical points of that function. Three types of critical points exist: At a local minimum a closed surface component is created. At a saddle either the *genus* of an isosurface changes, *i.e.*, holes appear or disappear in a surface component, or surface components separate or merge. At a maximum a closed surface component vanishes. Classical Morse theory classifies any given position of a data set by examining a small neighborhood around it. Considering a C^2-continuous function f, critical points can be determined analytically as points where the gradient ∇f vanishes. An associated type of the critical point is determined by the signs of the eigenvalues of the Hessian at that point.

I consider the common case of data sets with data values given at vertices of a uniformly spaced rectilinear grid and define isosurface topology by assuming that trilinear interpolation is used within individual cells. The topology of the level set of the trilinear interpolant in each cell is used to define the level set topology for the whole data set. For general C^0-continuous functions, including those given by piecewise trilinear interpolation (at mesh vertices, on mesh edges and on mesh faces), gradient and Hessian are undefined. Based on the work of Banchoff [4], who

observed that critical points are intrinsic geometrical in nature, it is possible to "emulate" these criteria for discrete data sets by looking at a neighborhood around a point and checking how many regions with a larger and how many regions with a smaller value than the value at the considered location exist. This definition has been used in computational geometry to detect critical points of piecewise linear scalar fields defined on simplicial complexes, see Edelsbrunner *et al.* [18, 19] and Gerstner and Pajarola [28].

Detected critical points are utilized to aid a user in data exploration with isosurfaces and for automatic transfer function design for direct volume rendering. An "isosurface navigator" lists all critical points along with their type. When a critical point is selected, its location is marked in the data set. It is possible to center the view on the corresponding critical point and set the currently considered isovalue to a value slightly below, equal to, or slightly above the critical value. Critical isovalues are also used to generate transfer functions using schemes presented by Fujishiro *et al.* [24, 26]. Resulting transfer functions either emphasize on zones of equal topological behavior or values close to critical isovalues.

To handle general data sets it is necessary to extend the concept of critical points to critical regions. While Morse theory requires that only isolated points can be critical, it is possible that a data set contains entire regions that are critical: A torus can form around a circle constituting a minimum, a new component can appear around a constant value sub-volume, or two surfaces can merge along a line denoting a saddle. By considering the neighborhood around a region instead of a point, I extend the concept of criticalities to regions in a data set. This allows me to capture isosurface topology changes in data sets that contain regions of constant value.

The remainder of this chapter proceeds as follows. After motivating the use of the topology of piecewise trilinear interpolation as reference topology, a definition of critical points that can be used for piecewise trilinear interpolation is introduced. Subsequently, a method is presented that detects critical points for data given on a rectilinear grid when piecewise trilinear interpolation is used and the requirement that two edge-connected vertices differ in value holds. Subsequently, the original definition is extended to consider and classify regions of constant value. This allows to discard the requirement that edge-connected vertices must differ in value and handle a wider variety of volumetric data sets.

6.2 Topology Definition for Samples on Rectilinear Grids

When considering data on a rectilinear grid, values within cells are obtained by interpolation. Different choices of interpolants lead to different contour topologies. The topology of piecewise trilinear interpolation was chosen as "reference topology." Marching cubes (MC) is a de-facto standard for isosurface extraction on rectilinear grids. My goal is to provide information about an isosurface extracted by the MC method. In the course of this work I considered different variants of the MC algorithm, see Section 2.2.5. The original MC algorithm could produce cracks in an extracted isosurface triangulation. Today, implicit disambiguation is the most commonly used MC variant, as it only requires minor modifications to the MC case table. Using "implicit disambiguation" leads to problems when detecting critical isovalues. By "merging" minima and maxima at vertices with saddles on boundary faces and in a cell's interior it causes "discontinu-

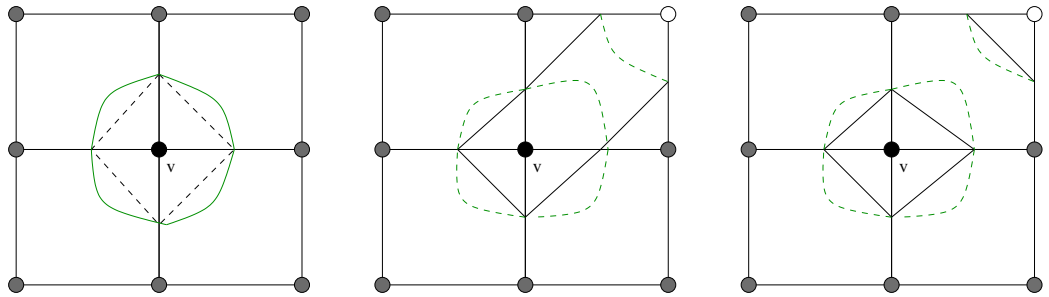


Figure 6.1: Bilinear interpolation. A vertex v is a minimum, if its four edge-connected neighbors have larger values (gray). (a) If no saddle within a face exists, this is correctly determined by an unmodified MC method. (b) If a saddle exists within a face and "implicit disambiguation" always separates positive vertices, a topologically incorrect isosurface results. The minimum is no longer origin of a new, connected component. It is "merged" with the saddle. (c) The asymptotic decider extracts a correct isosurface that preserves the minimum.

ous" topology changes. This necessitates to consider more than the neighboring, edge-connected vertices to classify a vertex of the grid, see Figure 6.1.

The assumption of MC, that an edge is intersected once if and only if its vertices differ in polarity, remains sound. Essentially, this assumption states that the sampling rate of the grid is sufficiently high to capture an isosurface. Using linear interpolation to determine an edge intersection point with the isosurface is an obvious choice. Bilinear interpolation on cell faces and trilinear interpolation within a cell are natural extensions of linear interpolation. Trilinear interpolation is commonly used in scientific visualization. A variety of MC extensions exist that use contours of the trilinear interpolant to determine isosurface topology within a cell. Thus, using the topology of piecewise trilinear interpolation is a sensible choice for an critical point detection scheme. Furthermore, it allows to classify a vertex based on its edge-connected neighbors. Consequently, the critical isovalues detected are only meaningful if a MC scheme is used that extracts a topologically correct isosurface triangulation of a trilinear function.

An alternative to trilinear interpolation is computing a tetrahedrization of the domain and using linear interpolation within tetrahedra. Figure 6.2 illustrates a problem arising from this procedure: Figures 6.2(a) and 6.2(b) show two possible triangulations of samples on a rectilinear, regular grid. In each figure, interior points are classified as ordinary point ("Or"), minimum ("Mi"), or saddle ("Sa"). Possible contours are shown as bold dashed lines. Contour topology and the classification of vertices depends on the chosen, arbitrary triangulation. Figure 6.2(c) shows the same classification for piecewise bilinear interpolation. Saddles may exist in the interior of cells, leading to "smooth" transitions between contour topologies.

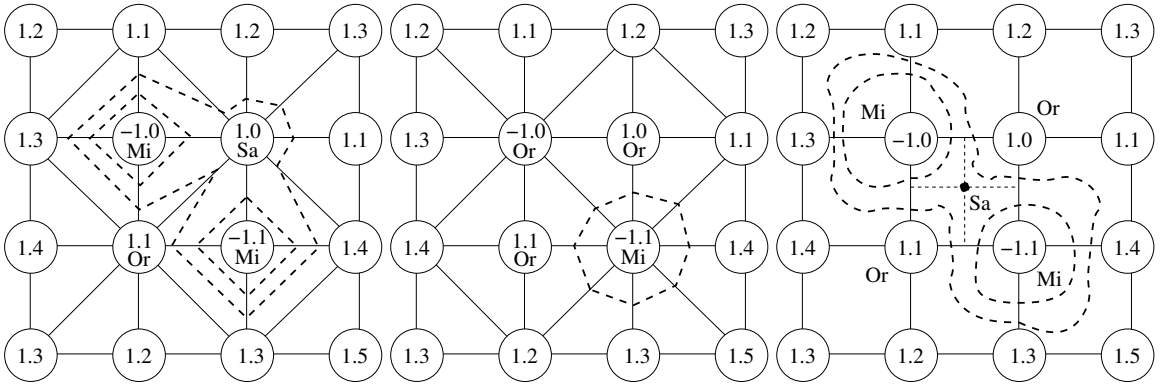


Figure 6.2: (a), (b) When a rectilinear grid is triangulated, contour topology and critical points depend on the choice of the triangulation. (c) Topology of trilinear interpolation.

6.3 Detecting Critical Points

The goal of this work is to detect *critical isovalues* of a piecewise trilinear scalar field given on a regular rectilinear grid. Gerstner and Pajarola [28] developed criteria for detecting critical points of piecewise linear scalar fields defined on tetrahedral meshes and used them in mesh simplification. These criteria are related to the work of Banchoff [4] and Edelsbrunner *et al.* [18, 19], see Section 2.1.2. I provide a comprehensive analysis of the topological behavior of piecewise trilinear interpolation and develop criteria to detect critical isovalues for these scalar fields. I further develop methods to use these critical isovalues for volume data exploration.

6.3.1 Definitions

For a C^2-continuous function f, critical points occur where the gradient ∇f assumes a value of zero, *i.e.*, $\nabla f = 0$. The type of a critical point $\mathbf{p}$ is determined by the signs of the eigenvalues of the Hessian of f at $\mathbf{x}$. Piecewise trilinear interpolation when applied to rectilinear grids, in general, produces only C^0-continuous functions. Therefore, it is necessary to define critical points differently. This is possible using a criterion based on the work of Banchoff [4] and Edelsbrunner *et al.* [18, 19], see also Section 2.1.2. Instead of considering upper and lower star of a triangulation, a neighborhood around a point is examined and partitioned into "positive" and "negative" regions (corresponding to the upper and lower star), leading to the following definition.

Definition 5 (Regular and Critical Points) *Let $F : \mathbb{R}^d \to \mathbb{R}$, $d \geq 2$, be a continuous function. A point $x \in \mathbb{R}^d$ is called a (a) regular point, (b) minimum, (c) maximum, (d) saddle, or (e) flat point of F, if for all $\epsilon > 0$ there exists a neighborhood $U \subset U_\epsilon$ with the following properties: If $\dot{\bigcup}_{i=1}^{n_p} P_i$ is a partition of the preimage of $[F(s), +\infty)$ in $U - \{x\}$ into "positive" connected components and $\dot{\bigcup}_{j=j}^{n_n} N_j$ is a partition of the preimage of $(-\infty, F(s))]$ in $U - \{x\}$ into "nega-*

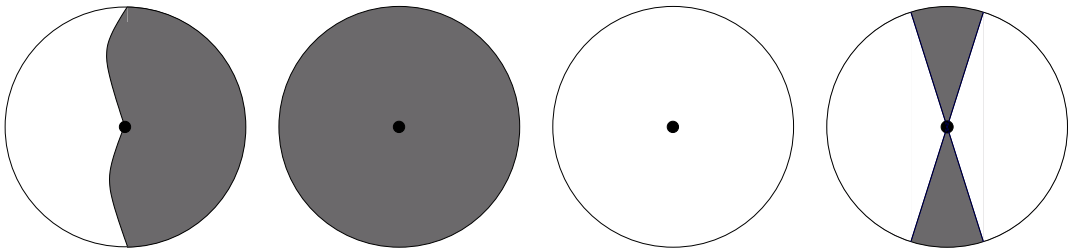


Figure 6.3: Classification of a point based on connected regions in its neighborhood: (a) Around a regular point $\mathbf{x} \in \mathbb{R}^3$, the isosurface $F^{-1}(F(x))$ divides space into a single connected volume P with $F > 0$ (gray) and a single connected volume N with $F < 0$ (white). (b) Around a minimum, every point in U has a larger value than $F(\mathbf{x})$. (c) Around a maximum, every point in U has a smaller value than $F(\mathbf{x})$. (d) In case of a saddle, there are more than one separated regions with values larger or smaller than the value $F(\mathbf{x})$.

tive" connected components, then (a) $n_p = n_n = 1$ *and* $P_1 \neq N_1$*, (b)* $n_p = 1$ *and* $n_n = 0$*, (c)* $n_n = 1$ *and* $n_p = 0$*, (d)* $n_p + n_n > 2$*, or (e)* $n_p = n_n = 1$ *and* $P_1 = N_1$*.*

Remark 1 *For (a) – (d), see Figure 6.3. Concerning case (e), all points in* U *have the same value as* $F(\mathbf{x})$*. It is possible to extend the concept of being critical to entire regions and classify regions rather than specific locations, see Section 6.6.*

Remark 2 *The cases* $n_p = 2$*,* $n_n = 0$ *and* $n_p = 0$*,* $n_n = 2$ *are not possible for* $d \geq 2$*.*

Piecewise trilinear interpolation reduces to bilinear interpolation on cell faces and to linear interpolation along cell edges. All values that trilinear interpolation assigns to positions in a cell lie between the minimal and maximal function values at the cell's vertices (convex hull property). In fact, maxima and minima can only occur at cell vertices. If two vertices connected by an edge have the same function value, the entire edge can represent an extremum or a saddle, see Section 6.6. It is even possible that a polyline defined by multiple edges in the grid, or a region consisting of several cells, becomes critical. In these cases, it is no longer possible to determine, locally, whether a function value is a critical isovalue. To avoid these types of problem, a restriction on the data is imposed requiring that function values at vertices connected by an edge must differ. Saddles can occur at cell vertices, on cell faces of a cell, and in a cell's interior, but not on cell edges. This fact is due to the restriction that an edge cannot have one constant function value.

Lemma 2 (Regular Edge Points) *Every point on an edge of a trilinear interpolant with distinct edge-connected values is a regular point.*

Proof: By assumption, the two endpoints of the edge have different values. Interpolation along edges is linear, and the derivative differs from zero. The implicit function theorem defines neighborhoods $U_i \times V_i$ and a height function $h_i : U_i \Rightarrow V_i$ in each of the four cubes around the edge,

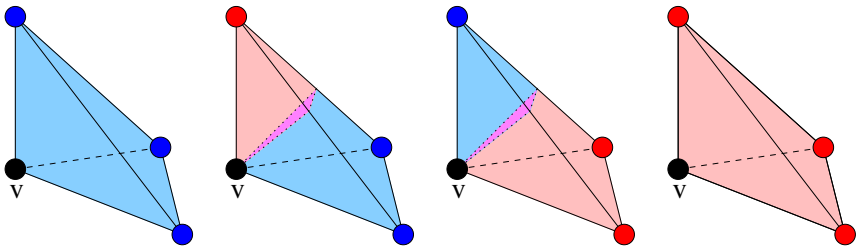


Figure 6.4: Possible partitions of a linear tetrahedron into connected positive and negative components.

such that the isosurface is a height field in the direction of the edge. Setting U to the smallest interval and determining suitable U_i defines a neighborhood such that the larger and smaller values are above and below a single height field. Therefore, a point on an edge is a regular point, because it is possible to start the construction with an arbitrary small neighborhood around $\mathbf{x}$. □

Thus, in order to detect critical isovalues of a piecewise trilinear scalar field, it is only necessary to detect critical values at vertices of a grid and saddle values within cells and on their boundary faces.

6.3.2 Critical Values at Vertices

Based on the work of Gerstner and Pajarola [28] a vertex is classified based on the polarity of its "surrounding" vertices. When values are obtained by piecewise linear interpolation applied to tetrahedral grids, critical points can only occur at mesh vertices. A mesh vertex can be classified based on its relationship with respect to vertices in a local neighborhood, *i.e.*, vertices connected to it via an edge. Classification is performed by constructing an "edge graph" whose connected components correspond to connected regions in a neighborhood around that vertex. Values in this neighborhood are defined by all linear tetrahedra sharing that vertex forming a "surrounding polyhedron." A linear tetrahedron consists of one negative, one negative and one positive, or one positive connected region, see Figure 6.4. Each of these regions contains at least one vertex of the tetrahedron. Consequently, these regions can be "represented" by vertices of that tetrahedron. Since at most two regions of opposite polarity occupy a linear tetrahedron, two vertices of same polarity always belong to the same region. A graph representing the neighborhood can be constructed having the vertices of the surrounding polyhedron (without the classified vertex) as nodes. These vertices are marked with a "+" if their associated function values are greater than the value of the classified vertex; or they are marked with a "-" if their associated function values are less than the value of the classified vertex. Equal values are not considered. Edges of the surrounding polyhedron define an edge graph. In this graph, all edges connecting vertices of different polarities are deleted as their corresponding regions also differ in polarity and are never connected. Each connected component in the remaining graph consists of tetrahedron

vertices that belong to the same connected region in the neighborhood. Consequently, a vertex is classified according to the number of connected components in this graph. If this number is one, the classified vertex is a maximum or minimum (depending on the sign of the connected component). If it is two, the classified vertex is a regular point. Otherwise, the vertex is a saddle point.

In the remainder of this section, it is shown that, when restricted to a small neighborhood around a vertex, trilinear interpolation partitions that neighborhood in similar way as linear interpolation partitions tetrahedra. Consequently, it is possible to classify a vertex in a hexahedral mesh based on its edge-connected neighbors (*i.e.*, six surrounding vertices) by applying a criterion similar to the one used to classify a vertex in a tetrahedral mesh.

Lemma 3 (Local Maximum) *Consider a cell C with vertex numbering as shown in Figure 2.22. If $v_0 > \max\{v_1, v_2, v_4\}$, then v_0 is a local maximum in C.*

Proof: Choose $m := \max\{v_1 - v_0, v_2 - v_0, v_4 - v_0\} < 0$ and $M := \max\{v_3 - v_0, v_5 - v_0, v_6 - v_0, v_7 - v_0, 1\} \geq 1$. Let $v_i' = v_i - v_0$. If $0 < x, y, z < \epsilon := \frac{|m|}{3|M|}$, then

$$
\begin{aligned}
F(x,y,z) - v_0 =& (1-x)(1-y)(1-z)v_0' + x(1-y)(1-z)v_1' + (1-x)y(1-z)v_2' + \\
& xy(1-z)v_3' + (1-x)(1-y)zv_4' + x(1-y)zv_5' + (1-x)yzv_6' + xyzv_7' \\
<& x(1-y)(1-z)m + (1-x)y(1-z)M + xy(1-z)m + \\
& (1-x)(1-y)zM + x(1-y)zm + (1-x)yzM + xyzM \\
\leq& m\epsilon\left[(1-y)(1-z) + (1-x)(1-z) + (1-x)(1-y)\right] + \\
& M\epsilon^2\left[1 - z + 1 - x + z + 1 - y\right] \\
\leq& 3m\epsilon(1-\epsilon)^2 + 3M\epsilon^2 \\
=& 3\frac{|m|}{3|M|}\left(\operatorname{sgn}(m)|m|\left(1 - \frac{|m|}{3|M|}\right)^2 + M\frac{|m|}{3|M|}\right) \\
=& \frac{|m|}{|M|}\left(-|m| + \frac{2}{3}\frac{|m|}{|M|} - \frac{|m|^2}{9|M|^2} + M\frac{|m|}{3|M|}\right) \\
=& \frac{|m|}{|M|}\left(-\frac{2}{3}|m| + \frac{2}{3}\frac{|m|}{|M|} - \frac{|m|^2}{9|M|^2}\right) \\
\leq& \frac{|m|}{|M|}\left(-9\frac{|m|^2}{|M|^2}\right) < 0.
\end{aligned}
$$

□

Lemma 4 (Linear Cell Partition) *Consider a cell C with vertex values v_i and vertex positions $\mathbf{p}_i$ numbered as shown in Figure 2.22. If $v := v_0 \neq v_1, v \neq v_2, v \neq v_4$ holds, then for all $\epsilon > 0$ there exists a $\delta < \epsilon$ such that for the intersection $R = U_\delta(\mathbf{p}_0) \cap C$ the following statements hold: (a) If $v > \max\{v_1, v_2, v_4\}$ then $n_n = 1$ and $N_1 = R$, i.e., all values in the region are less than v. (b) If there exist $i, j, k \in \{1, 2, 4\},\ i \neq j \neq k,\ i \neq k$, such that $v > \max\{v_i, v_j\}$ and $v < v_k$, then $n_n = n_p = 1$ and R completely contains a surface dividing N_1 and P_1. Furthermore, all values on the triangle $\mathbf{p}_0\mathbf{p}_i\mathbf{p}_j$ are less than v. (c) If there exist $i, j, k \in \{1, 2, 4\},\ i \neq j \neq k,\ i \neq k$, such that $v < \min\{v_i, v_j\}$ and $v > v_k$, then $n_n = n_p = 1$, and R completely contains a surface*

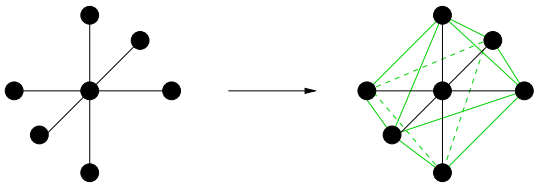


Figure 6.5: Edge-connected vertices as part of an implicit tetrahedrization.

dividing N_1 and P_1. Furthermore, all values on the triangle $\mathbf{p}_0\mathbf{p}_i\mathbf{p}_j$ are less than v. (d) If $v < \max\{v_1, v_2, v_4\}$, then $n_n = 1$ and $N_1 = R$, i.e., all values in the region are greater than v.

Proof: Cases (a) and (d) are symmetrical and follow from Lemma 3. Cases (b) and (c) are symmetrical as well, and it is sufficient to prove one of them. Similarly, the same holds when any other v_i is chosen as v and its edge-connected neighbor vertices are considered.

Let $\epsilon > 0$. The derivative of F at $\mathbf{p}_0$ is $(v_1 - v_0, v_2 - v_0, v_4 - v_0)$. There exists an $\epsilon > \delta > 0$ such that the derivative has rank 1 in the whole neighborhood $R = U_\delta(\mathbf{p}_0) \cap C$. In this case, the regular value theorem guarantees the existence of an isosurface with function value v_0 dividing $U_\delta(\mathbf{p}_0)$ into a single region with larger and a single region with lower function values. If the surface intersects C outside $\mathbf{p_0}$, R is split into exactly two parts. If not, $\mathbf{p_0}$ is a local maximum or minimum. This fact proves the first part of (b) and (c). For small $\epsilon > \delta > 0$, a calculation similar to the proof of Lemma 3 demonstrates that the face with $\mathbf{p}_0, \mathbf{p}_i, \mathbf{p}_j$ is not intersected inside R by the isosurface in cases (b) and (c). □

Using the L_1-norm[1], the intersection of a neighborhood with a cell corresponds to a tetrahedron. According to Lemma 4, this tetrahedron is partitioned in the same way as a tetrahedron using linear interpolation (even when, as in this case, partitioning surfaces are not necessarily planar), see Figure 6.4. A vertex can be classified by considering its edge-connected neighbor vertices. These vertices are treated as part of a local implicit tetrahedrization surrounding a classified vertex, where the classified vertex and three edge-connected vertices belonging to the same rectilinear cell imply a tetrahedron, see Figure 6.5. Using the criterion developed for linear on tetrahedral meshes, a vertex can be classified by considering the implicit tetrahedrization of its neighborhood. Computing the connected components in an edge graph for all possible vertex polarity configuration using this implicit tetrahedrization, an LUT with $2^6 = 64$ entries is obtained that maps a configuration of "+" and "-" of edge-connected vertices to a vertex classification. I decided to generate this relatively small LUT manually.

6.3.3 Critical Values on Faces

When linear interpolation is used, critical points can only occur at grid vertices. When piecewise trilinear interpolation is used, critical points can also occur on boundary faces. On a boundary face piecewise trilinear interpolation reduces to bilinear interpolation and the interpolant on a face can have a saddle. This face saddle is not necessarily a saddle of the piecewise trilinear

[1] $\|x\|_1 = \sum_i |x_i|$

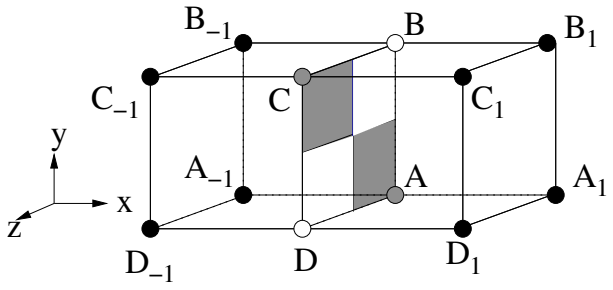


Figure 6.6: Vertex numbering scheme used in Lemma 5.

interpolant. The following lemma provides a criterion to whether a face saddle is a saddle of the trilinear interpolant:

Lemma 5 (Face Saddle) *Let* $\mathbf{p}$ *be a point on the shared face of two cells, where both trilinear interpolants degenerate to the same bilinear interpolant. The point* $\mathbf{p}$ *is a saddle point when these two statements hold:*

1. *The point* $\mathbf{p}$ *is a saddle point of the bilinear interpolant defined on the face.*
2. *With the notations of Figure 6.6, where, without loss of generality, the saddle value is zero, and cells are rotated such that* A *and* C *are larger than zero* $connected_1$ *and* $connected_2$ *both differ from zero and have the same sign.*

$$connected_1 = \begin{cases} c_{1.1} & \text{if } c_{1.1} \neq 0 \\ c_{1.2} & \text{if } c_{1.1} = 0 \end{cases} \text{ , with} \tag{6.1}$$
$$c_{1.1} = C(A_1 - A) + A(C_1 - C) - D(B_1 - B) - B(D_1 - D) \text{ , and}$$
$$c_{1.2} = (A_1 - A)(C_1 - C) - (B_1 - B)(D_1 - D) \ .$$

$$connected_2 = \begin{cases} c_{2.1} & \text{if } c_{2.1} \neq 0 \\ c_{2.2} & \text{if } c_{2.1} = 0 \end{cases} \text{ , with} \tag{6.2}$$
$$c_{2.1} = C(A_{-1} - A) + A(C_{-1} - C) - D(B_{-1} - B) - B(D_{-1} - D) \text{ , and}$$
$$c_{2.2} = (A_{-1} - A)(C_{-1} - C) - (B_{-1} - B)(D_{-1} - D) \ .$$

Otherwise, $\mathbf{p}$ *is a regular point of the trilinear interpolant.*

Proof:

1. If $\mathbf{p}$ is not a saddle of the bilinear interpolant on the face, one partial derivative on the face is different from zero. The regular value theorem implies the existence of a dividing isosurface in both cells in a small neighborhood $U_\delta(\mathbf{p}) \subset U_\epsilon(\mathbf{p})$, leading to a single isosurface in the whole neighborhood that splits into one connected component with values larger than $f(\mathbf{p})$ and one connected component smaller than $f(\mathbf{p})$. Consequently, $\mathbf{p}$ is a regular point of piecewise trilinear interpolation.

2. Let $\mathbf{p}$ be a saddle point with respect to the bilinear interpolant on the face. (This proof adopts an idea from Chernyaev [11].) Without loss of generality, assume that the saddle value is zero. If the saddle value differs from zero, it can be subtracted from all vertex values reducing this case to the case of the saddle value being zero. To simplify notation, assume that the face is perpendicular to the x-coordinate axis. If any slice $x = \text{const}$, $x \in [0, 1]$, parallel to the face, is considered the function F becomes

$$\begin{aligned} F_x(y,z) &= A_x(1-y)(1-z) + B_x y(1-z) + C_x yz + D_x(1-y)z \text{ , with} \\ A_x &= A(1-x) + A_1 x \text{ ,} \\ B_x &= B(1-x) + B_1 x \text{ ,} \\ C_x &= C(1-x) + C_1 x \text{ , and} \\ D_x &= D(1-x) + D_1 x \text{ .} \end{aligned} \tag{6.3}$$

As pointed out in Section 2.2.5, the sign of the saddle value at the intersection of the asymptotes

$$SV(x) = \frac{A_x C_x - B_x D_x}{A_x + C_x - B_x - D_x} \tag{6.4}$$

determines whether positive or negative regions are connected on that slice. By the choice of "cell rotation," A and C are larger than zero. Since the boundary face contains a saddle with a value of zero within its boundaries, B and D must be smaller than zero. Consequently, $A + C - B - D$ is always positive. When bilinear slices are considered that are sufficiently close to the considered cell face, *i.e.*, for small values of x, the same holds for $A_x + C_x - B_x - D_x$. Consequently, it is sufficient to consider the sign of the denominator $A_x C_x - B_x D_x$ of Equation 6.4 to determine which regions are connected on the slice. For $x = 0$, this expression is zero since the saddle value is computed as

$$\frac{AC - BD}{A + C - B - D} = 0 \quad \Rightarrow \quad AC - BD = 0$$

is zero. For an arbitrary slice at distance x from the face, the value of the denominator can be expressed as

$$\begin{aligned} Den(x) =& ax^2 + bx + c \text{ , with} \\ a =& (A_1 - A)(C_1 - C) - (B_1 - B)(D_1 - D) \text{ ,} \\ b =& C(A_1 - A) + A(C_1 - C) - D(B_1 - B) - B(D_1 - D) \text{ , and} \\ c =& AC - BD = 0 \text{ .} \end{aligned} \tag{6.5}$$

Computing the first derivative of $A_x C_x - B_x D_x$ with respect to x at $\mathbf{p}$, *i.e.*, for $x = 0$, which turns out to be $f'(0) = 2a\dot{0} + b = C(A_1 - A) + A(C_1 - C) - D(B_1 - B) - B(D_1 - D)$, it is possible to determine whether $A_x C_x - B_x D_x$ is positive or negative above $\mathbf{p}$. If it is positive, the negative regions are connected above $\mathbf{p}$. If it is negative, the positive regions are connected above $\mathbf{p}$. If the first derivative is zero, the sign of the second derivative for $x = 0$ can be used, which is $f''(0) = 2a = 2(A_1 - A)(C_1 - C) - (B_1 - B)(D_1 - D)$ (which

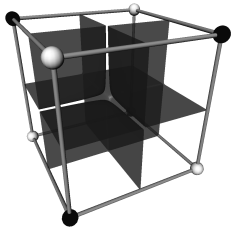


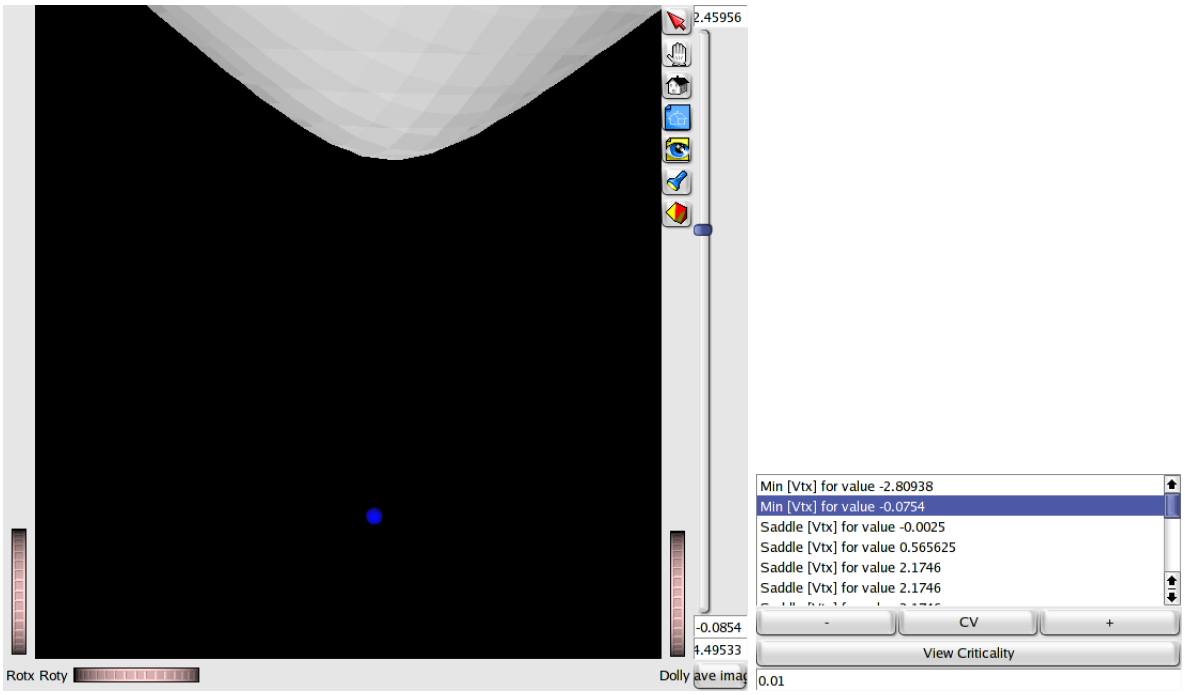
Figure 6.7: In some cases, face saddles can be connected within a cell's interior. The saddle is no longer localized and isosurface components meet along straight lines. In the figure, isosurface components meet along axis-aligned lines. (Vertex values are $v_0 = 1$, $v_1 = -1$, $v_2 = 1$, $v_3 = -1$, $v_4 = -1$, $v_5 = 1$, $v_6 = -1$, and $v_7 = 1$)

has the same sign as $(A_1 - A)(C_1 - C) - (B_1 - B)(D_1 - D)$). If it is positive, the positive regions are connected above $\mathbf{p}$. If it is negative, the negative regions are connected above zero. All higher derivatives are zero. If first and second derivative are both zero, it follows that $a = 0$, $b = 0$ and $c = 0$. Consequently, $Den(x) = 0$ holds, *i.e.*, the denominator is zero for locations within the cell. If slices close to the face containing the saddle are considered, the nominator differs from zero (because it is positive). This implies that the saddle value is zero for this slices. Consequently, a saddle with the same value as on the considered face exists on these slices. The considered saddle is not localized to a point and no critical point in the classical sense. The final criterion results from application of this idea to both cells sharing the face. If the negative or positive values are connected around $\mathbf{p}$ in both cubes, $\mathbf{p}$ is a saddle of the piecewise trilinear interpolant, otherwise $\mathbf{p}$ is not a critical point of the piecewise trilinear interpolant. □

Thus, face saddles of piecewise trilinear interpolation can be detected effectively by considering all cell faces for a saddle of the bilinear interpolants on faces and checking whether the criterion stated in Lemma 5 holds. If either $connected_1$ or $connected_2$ is zero, a non-localized saddle occurs in the corresponding cell. Figure 6.7 shows an example of a cell where isosurface components meet along axis-perpendicular lines. Strictly speaking, no critical point occurs. However, it can be useful to permit and consider extended criticalities, see Section 6.6. However, in these cases it is no longer possible to classify a saddle on a boundary faces based on the two adjacent cell. Instead, the saddle must be traced through the cell, and additional cells must be considered, see Section 6.9.1.

6.3.4 Critical Values inside a Cell

Saddles of the trilinear interpolant in the interior of a cell are easy to handle as they are always saddles of the piecewise trilinear interpolant as well. Interior saddles are already used by various MC variants to determine isosurface topology within a cell. These saddles are computed using the equations given by Nielson [67], see Equation 2.46 in Section 2.2.5. Inner saddles of a trilinear interpolant that coincide with a cell's boundary faces or vertices are not necessarily saddles



(a) (b)

Figure 6.8: The "critical isovalue navigator" (b) allows to explore a volumetric data set using critical isovalues. Resulting isosurfaces are displayed in the "isosurface viewer."

of a piecewise trilinear interpolant. Trilinear interpolation assigns constant values to locations along coordinate-axis-parallel lines passing through the saddle. Currently, the possibility of an internal saddle coinciding with a vertex or an edge is ruled out. Otherwise, the requirement that edge-connected vertices differ in value would be violated. Saddles of trilinear interpolants that coincide with cell faces are also saddles of the bilinear interpolant on the face. As such they are discussed in Section 6.3.3.

6.4 Data Exploration Using Critical Points and Isovalues

A convenient way to use critical isovalues is to provide a user with a navigational tool in addition to an isosurface viewer. Prior to starting an isosurface viewer, critical isovalues are computed and displayed in an "isovalue navigator," see Figure 6.8. In this window, critical isovalues are listed along with a corresponding type (minimum, maximum, saddle, face saddle, interior saddle). When a user selects a critical isovalue, its corresponding critical point is marked by a sphere whose color depends on the type (blue, red and green representing a minimum, a maximum or a saddle, respectively). Buttons allow a user to set the isovalue of a displayed isosurface to a value slightly below, equal to, or slightly above a critical isovalue. The isovalue offset for the isosurfaces below and above a chosen critical isovalue is specified in a text field.

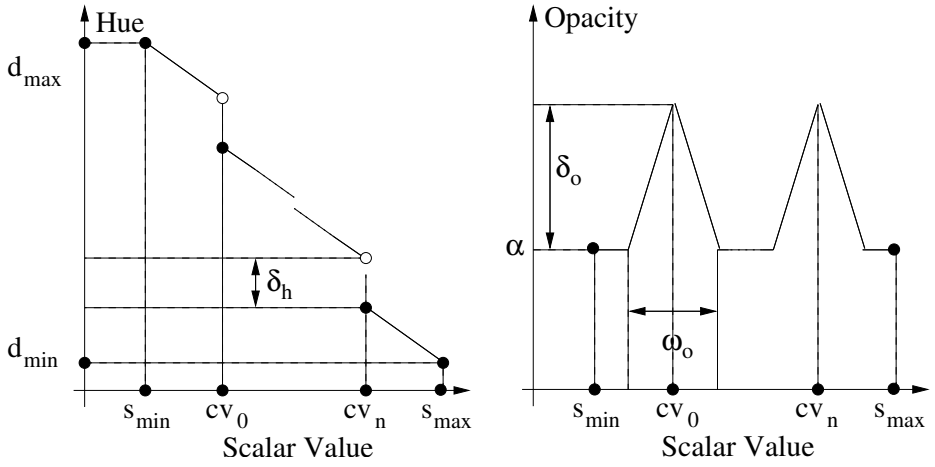


Figure 6.9: Transfer function emphasizing topologically equivalent regions.

In data sets containing several "nested isosurfaces," *i.e.*, data sets where one isosurface component is completely contained within another, it can be difficult to locate a critical point, even if its position is marked. The "isovalue navigator" contains a button that positions the camera so that the viewing focus is the critical point.

Isosurfaces are extracted using a MC approach based on the works of Lopes [55] and Nielson [67]. Lopes' MC approach is extended to correctly handle tunnels for Configuration $13b$, where two different tunnel connections are possible, see Section 2.2.5. (This extension predates Lopes and Brodlie's [54] correction of Lopes' method.) Lopes' original approach correctly detects the tunnel, but connects three boundary polygons to the tangent points, resulting in an invalid triangulation. By using Nielson's criterion to distinguish between the two sub-cases, it is possible to correct this flaw. Nielson's method uses the sign of the trilinear interpolant in the cuboid containing $DeV[T]$ to determine whether the positive or negative vertices must be connected. To correct Lopes' original approach, a flag is added to the LUT that specifies for each boundary polygon whether it is used only for a tunnel connecting positive or negative vertices. Since problems only arise in case $13j$, this flag can also have a value of "irrelevant," indicating that a boundary polygon is part of both tunnel types. If a tunnel is detected by the presence of six tangent points, its type is determined by Nielson's criterion. Only those boundary polygons are connected to the polyline formed by the tangent points whose flags indicate that they are part of that particular tunnel type. Nielson [67] explicitly computes DeVella's necklace by considering bilinear contours on axis-perpendicular slices, finding slices where the contour degenerates to a pair of axis-aligned asymptotes and using the intersection points of these asymptotes as points of DeVella's necklace. Since the points of DeVella's necklace are identical to Lopes' tangent points, Nielson's equations are used instead of Lopes' resulting in a more concise computation scheme.

Critical isovalues can also help in automatic transfer function design. Given a list of critical isovalues, a corresponding transfer function is constructed based on the methods described by Fujishiro *et al.* [26]. The domain of the transfer function corresponds to the range of scalar

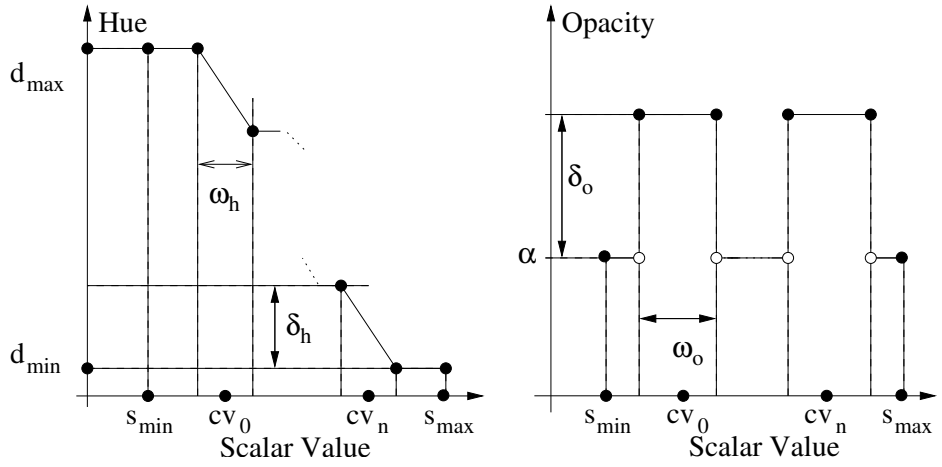


Figure 6.10: Transfer function emphasizing details close to critical isovalues.

values $[s_{min}, s_{max}]$ occurring in a data set. Outside this range the transfer function is undefined. Given a list of critical isovalues cv_i, either a transfer function emphasizing volumes containing topologically equivalent isosurfaces or a transfer function emphasizing structures close to critical values is constructed.

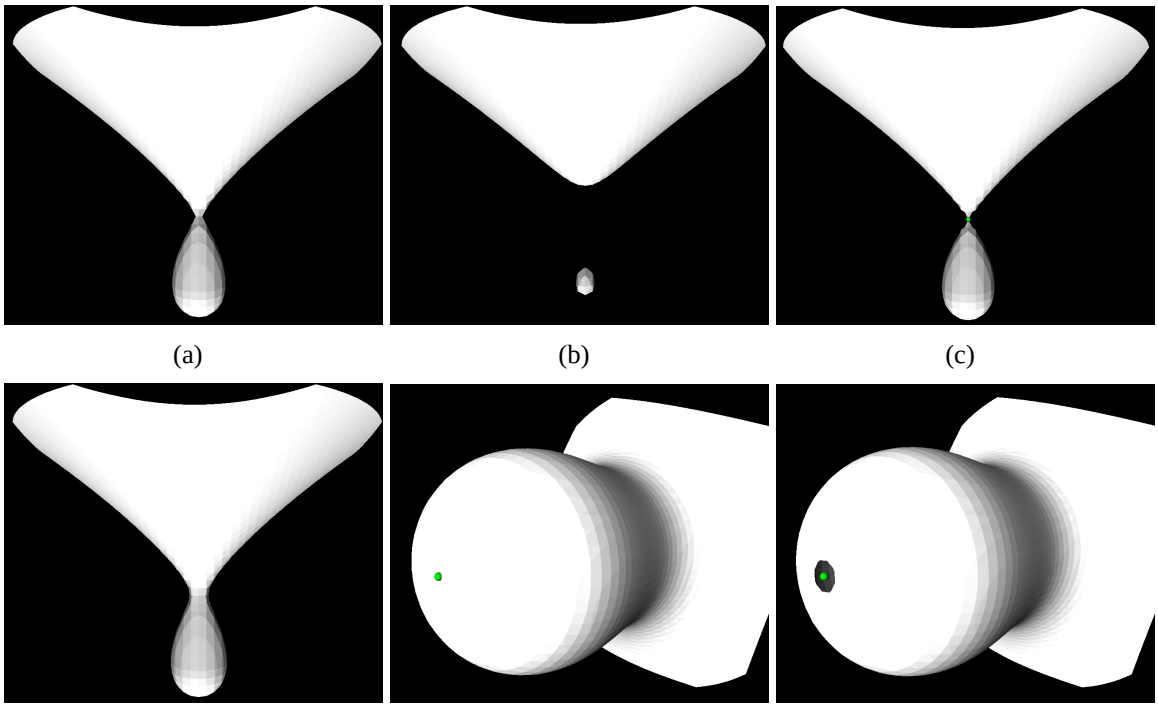
Figure 6.9 shows the construction of a transfer function that emphasizes on topologically equivalent regions. The color transfer is chosen such that hue uniformly decreases with the mapped value, except for a constant drop of δ_h at each critical value cv_i. The opacity is constant for all values except for hat-like elevations around each critical value cv_i having a width of ω_o and a height δ_o.

Figure 6.10 shows the construction of a transfer function emphasizing details close to critical isovalues. The hue transfer function is constant except for linear descents of a fixed amount δ_h within an interval with a width ω_h centered around each critical isovalue cv_i. The opacity is constant for all values except in intervals with a width ω_o centered around critical isovalues cv_i where the opacity is elevated by δ_o.

When several isovalues are so close together that intervals with a width ω_h or ω_o would overlap, all isovalues except the first are discarded to avoid high frequencies in the transfer function that could cause aliasing artifacts in the rendered image.

6.5 Results of Critical Point Analysis

Figure 6.11 shows the "Drip" data set, obtained by sampling the analytic function $F(x, y, z) = x^2 + y^2 - 0.5(0.995z^2 + 0.005 - z^3)$. (This function was provided by Terry J. Ligocki, Lawrence Berkeley National Laboratory.) The function was evaluated for $x, y, z \in [-1.5, 1.5]$, sampled on a 40^3 uniform rectilinear grid. Figure 6.11(a) shows the isosurface $F = 0$. My method can be used to detect critical values of this scalar field and show how this "drop" evolves. Originally, there exists just one component evolving from the boundary of the domain of the scalar field. An



(a) (b) (c)

(d) (e) (f)

Figure 6.11: Exploration of "Drip" data set.

"inner minimum" exists for a value of -0.0754. Figure 6.11(b) shows an isosurface for a value of $-0.0754 + 0.01$, where the isosurface component around the minimum already has grown to a visible size. An inner saddle of the scalar field exists for a value of -0.0025, shown in Figure 6.11(c), where a small green sphere marks the saddle position. The isosurface components are still distinct, but touch at the saddle position. For an isovalue of $-0.0025 + 0.01$, *i.e.*, a value slightly above the saddle value, both components have merged. Additional critical points arise on faces lying on the domain boundary. Figures 6.11(e) and 6.11(f) show a saddle that occurs as a result of the isosurface intersecting the domain boundary. A hole in the isosurface is clearly visible in Figure 6.11(f) as a result of domain boundary intersection. In this example, critical vertices on boundaries are detected by obtaining "missing" neighbor vertices by "mirroring" the other neighbor vertex in that direction. While this strategy correctly classifies minima and maxima on the grid boundary, it can lead to the detection of saddles where, in fact, none exist, see Figure 6.12. The solution is to generate independent case tables for all possible configurations, *i.e.*, classified vertex lying on a grid boundary face, a boundary edge, or coinciding with a grid corner. I did not implement this, as critical points on the mesh boundary can be viewed as "sampling" artifacts where the isosurface intersects the boundary of the domain. The neighborhood of a vertex is partially outside the domain.

Figure 6.13 shows a data set obtained by simulating a two-body distribution probability of a nucleon in the atomic nucleus "16O" when a second nucleon is known to be positioned at

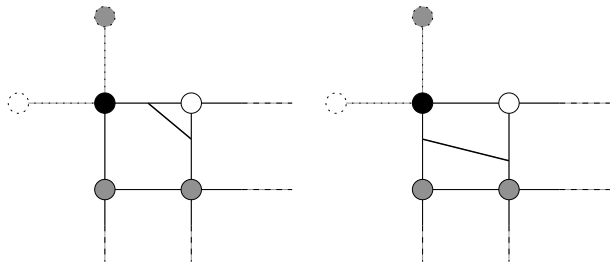


Figure 6.12: Mirroring vertices to classify boundary vertices can lead to the detection of saddles at locations where, in fact, none exist. The figure shows the classification of a vertex (shown as solid black disc) coinciding with the "corner" of the grid defining the domain. "Missing" neighbor vertices are obtained by mirroring the other neighbor vertex in the corresponding direction. The resulting "ghost" vertices are denoted by dotted lines. Vertices having a value larger than the classified vertex are colored gray. Vertices having a value smaller than the classified vertex are colored white. As a result of mirroring its neighbor the vertex is classified as saddle, while contour behavior suggests a regular point. Linear approximation of contours are shown as bold lines. Figure (a) shows a contour approximation for a value slightly below the value of the classified vertex. Figure (b) shows a contour approximation for a value slightly above the value of the classified vertex.

distance of 2 Fermi. This 41^3 data set is courtesy of the Sonderforschungsbereich (SFB) 382 of the German Research Council (DFG). It can be obtained at `http://www.volvis.org`. The isovalue navigator indicates a minimum for an isovalue of 19. From a greater distance, special contour behavior for this value cannot be perceived, see 6.13(a). By using the isovalue navigator to define a viewpoint close to the minimum and looking at the minimum, a second component forming inside the outer isosurface component becomes visible, see Figure 6.13(b). Several saddles exist. Among them is a saddle for the isovalue of 103, where one inner isosurface component merges with the outer component. A clipping plane is used in the figure to make the saddle location visible, see Figure 6.13(c). Figures 6.13(d) and 6.13(e) show a saddle inside the outer isosurface component. Several saddles exist for the same value; one of them can be seen in the background of the saddle, marked by a small green sphere.

Figure 6.14 shows the results of rendering the same data set with automatically generated transfer functions. Figure 6.14(a) emphasizes on volumes containing topologically equivalent isosurfaces. Details close to these critical isovalues are more visible in Figure 6.14(b).

Figure 6.15 shows the results of rendering a data set resulting from simulating the spatial probability distribution of the electrons in a high potential protein molecule. Figure 6.15(a) emphasizes on volumes containing topologically equivalent isosurfaces. Details close to these critical isovalues are better visible in Figure 6.15(b).

Figure 6.16 shows the results of rendering a data set resulting from a simulation of fuel injection into a combustion chamber. (Data set courtesy of SFB 382 of the German Research Council (DFG), see `http://www.volvis.org` for details.) Figure 6.16(a) emphasizes on

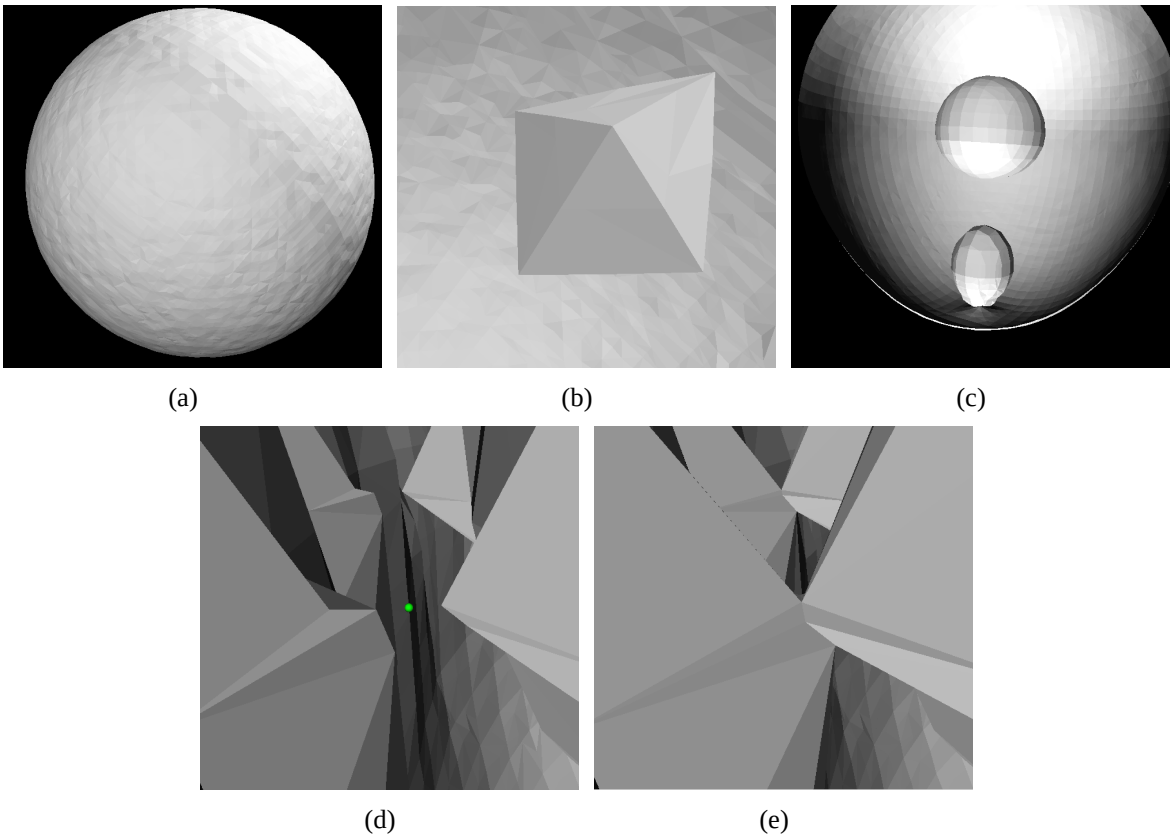
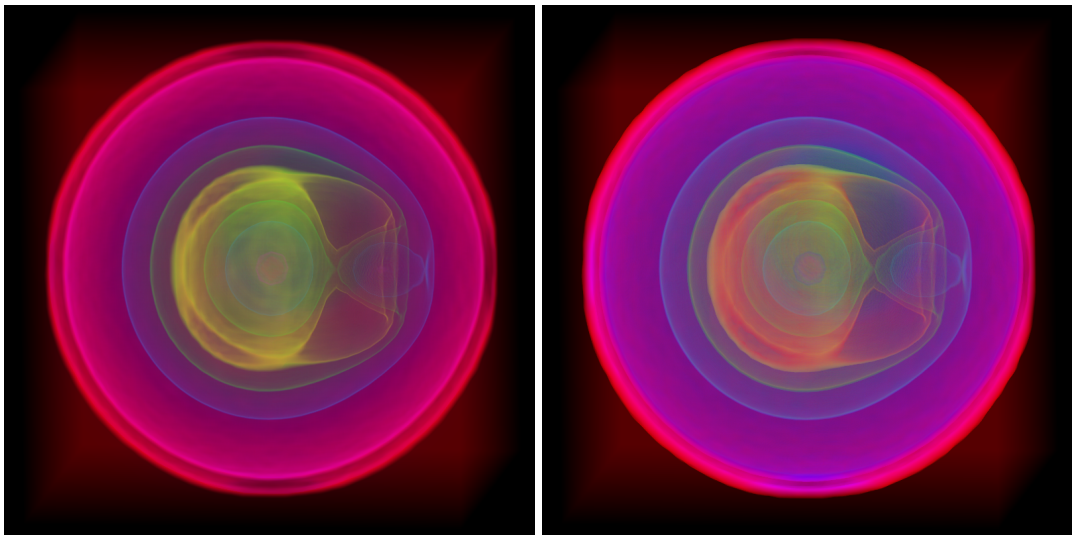


Figure 6.13: “Nucleon” data set. (Data set courtesy of SFB 382 of the German Research Council (DFG), see `http://www.volvis.org`)

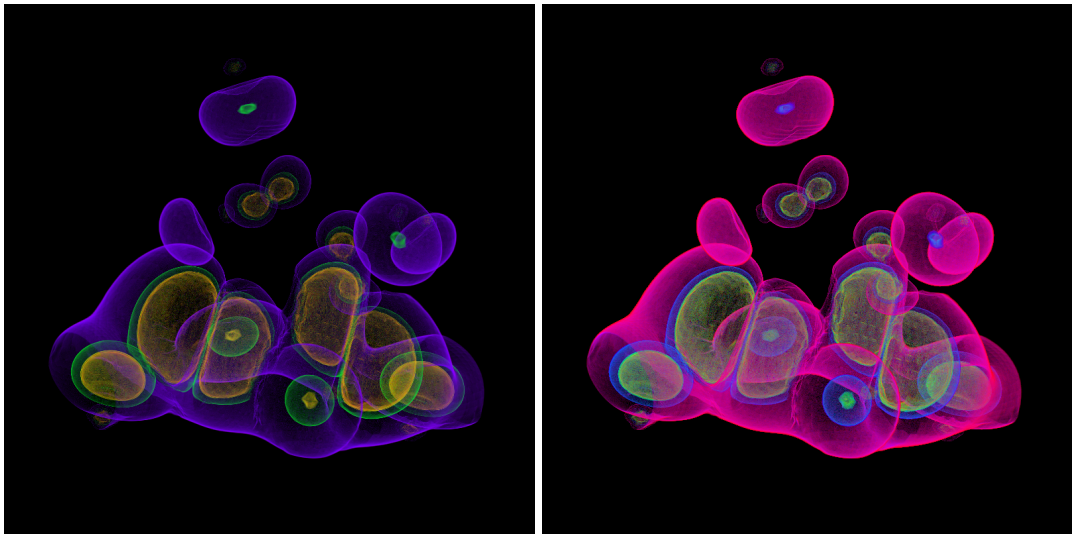
volumes containing topologically equivalent isosurfaces. Details close to these critical isovalues are more visible in Figure 6.16(b).



(a) Transfer function emphasizing topologically equivalent zones.

(b) Transfer function emphasizing structures close to critical isovalues.

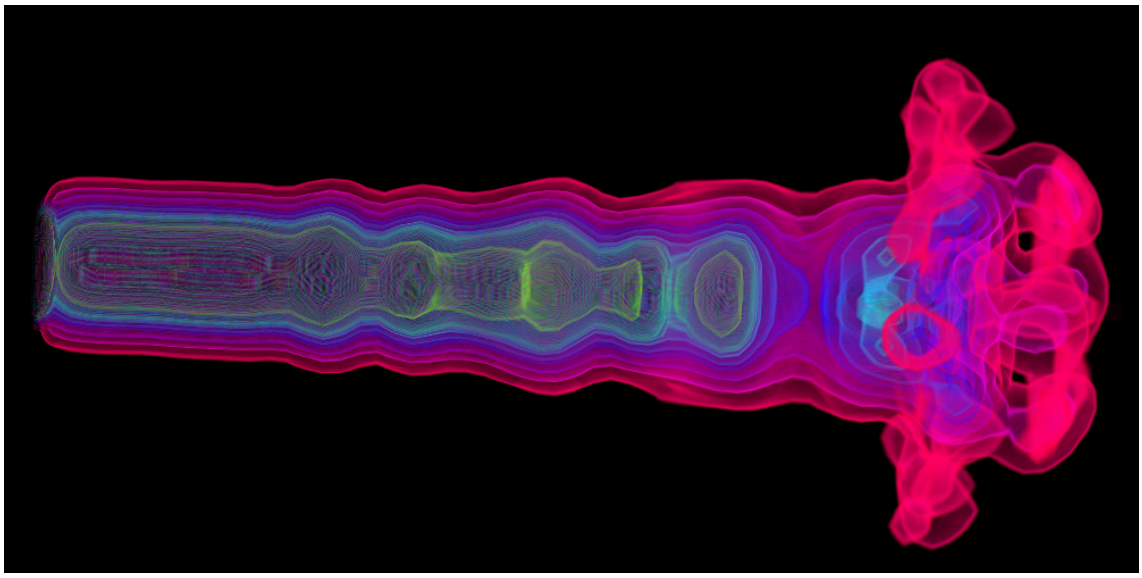
Figure 6.14: “Nucleon” data set. (Data set courtesy of SFB 382 of the German Research Council (DFG), see `http://www.volvis.org`)



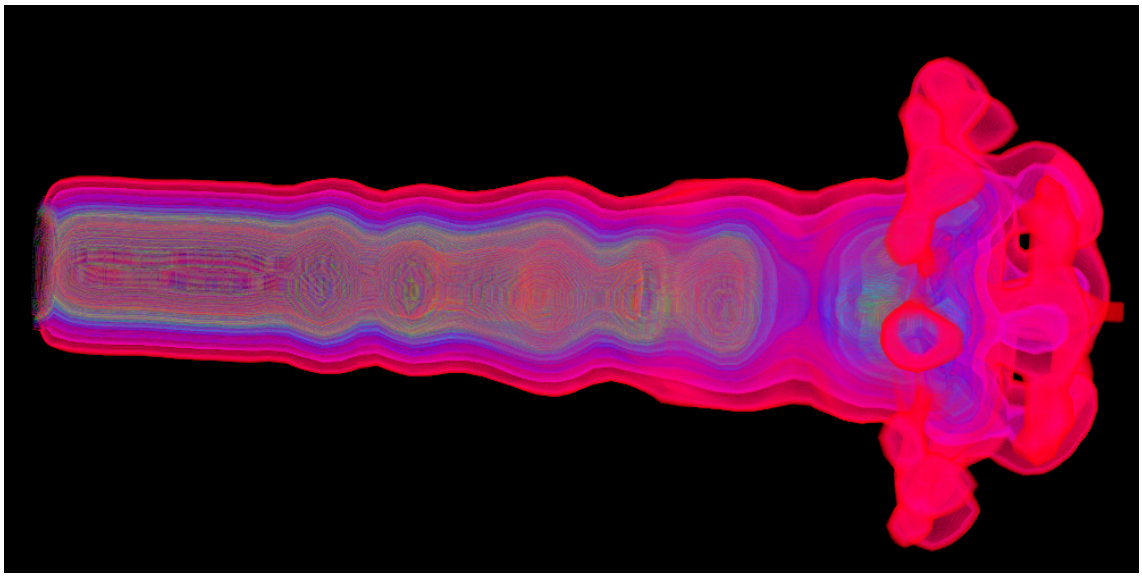
(a) Transfer function emphasizing topologically equivalent regions.

(b) Transfer function emphasizing structures close to critical isovalues.

Figure 6.15: “Neghip” data set. (Data set courtesy of VolVis Distribution of SUNY Stony Brook, NY, USA, see `http://www.volvis.org`)



(a) Transfer function emphasizing topologically equivalent regions.



(b) Transfer function emphasizing structures close to critical isovalues.

Figure 6.16: “Nucleon” data set. (Data set courtesy of SFB 382 of the German Research Council (DFG), see `http://www.volvis.org`)

6.6 From Critical Points to Critical Regions

To handle general data sets, it is further necessary to extend the concept of critical points to critical regions. While Morse theory requires that only isolated points can be critical, it is possible that a data set contains entire regions that are critical: A torus can form around a circle, which could be a minimum, a new component can appear around a constant value sub-volume, or two surfaces can merge along a line denoting a saddle. By extending the concept of isolated critical points to critical regions in a data set, it becomes possible to detect topology in data sets that contain regions of constant value.

In addition to a "flat region," see Definition 5 (e), it is possible to have "extended" maxima, minima, or saddles. Figure 6.7, for example, shows an extended saddle, where several isosurface components meet along axis-aligned lines. An extended minimum or maximum can be differentiated from its "localized" variant by the fact, that locations in its neighborhood exist that assume the same value as the considered locations. An extended saddle can be differentiated from a localized saddle by the fact that contours meet along an extended region instead of a single points. Definition 5 added locations with the same value as the classified point to both positive and negative regions. By modifying the definition to treat locations with the same value as the classified point independently, it becomes possible to differentiate between localized and extended critical points:

Definition 6 (Regular and Critical Points (extended)) *Let $M \subset \mathbb{R}^3$ be a mesh and $F : M \rightarrow \mathbb{R}$ be a C^0-continuous function that is C^∞-continuous in each grid cell. A point $\mathbf{x} \in \mathbb{R}^3$ is called (a) regular or ordinary, (b) minimum, (c) maximum, (d) saddle, (e) extended minimum, (f) extended maximum, (g) extended saddle, or (h) flat point of F, if for all $\epsilon > 0$ there exists a neighborhood $U \subset U_\epsilon(\mathbf{x})$ with the following properties: If $\dot{\bigcup}_{i=1}^{n_p} P_i$ is a partition of the preimage of $(F(\mathbf{x}), +\infty)$ in $U - \{\mathbf{x}\}$ into "positive" connected components, $\dot{\bigcup}_{j=j}^{n_n} N_j$ is a partition of the preimage of $(-\infty, F(\mathbf{x}))$ in $U - \{\mathbf{x}\}$ into "negative" connected components, and $\dot{\bigcup}_{k=1}^{n_z} Z_k$ is the partition of the preimage of $\{F(\mathbf{x})\}$ in $U - \{\mathbf{x}\}$ into "zero set" connected components, then (a) $n_p = n_n = n_z = 1$, (b) $n_p = 1$ and $n_n = n_z = 0$ (and U contains only ordinary points), (c) $n_n = 1$ and $n_p = n_z = 0$ (and U contains only ordinary points), (d) $n_p + n_n > 2$, $n_p, n_n \geq 1$, $n_z > 1$ (and U only contains ordinary points), (e) $n_p = 1$, $n_n = 0$, $n_z \geq 1$ (and U contains non-ordinary points), (f) $n_n = 1$, $n_p = 0$, $n_z \geq 1$ (and U contains non-ordinary points), (g) $n_p + n_n > 2$, $n_z = 1$ (and U contains non-ordinary points), and (h) $n_z = 1$ and $n_p = n_n = 0$.*

Remark 3 *The zero set corresponds to the level set for the value at $\mathbf{x}$.*

Remark 4 *Definition 6 is a modified form of Definition 5. The original definition added the zero set to both positive and negative sets. However, to extend the concept of critical points to critical regions it becomes necessary to consider the zero set individually.*

Remark 5 *Cases (a) – (d) describe localized critical and regular points and are illustrated in Figure 6.3. These definitions are equivalent to those used in topology and Morse theory. Cases (e) – (g) extend these concepts to extended regions. An extended minimum, maximum, or saddle*

corresponds to a region where each point in the region is a minimum, maximum, or saddle, respectively, according to Definition 5. Concerning case (h), all points in U have the same value as $F(\mathbf{x})$, i.e., the region has constant value.

Remark 6 *When U is small enough, the non-ordinary points inside U all have the same value $F(x)$: Inside a cell, this stems from the fact that the derivative is zero at a non-ordinary point and one can separate the regions with zero derivative. At faces, this fact holds since critical points on a face are also critical points for the bilinear interpolant, and the derivative of the bilinear interpolant is used to provide the answer. On an edge, it suffices to consider the linear interpolant, which leads to the same result. Finally, vertices of the grid have a neighborhood of points on edges, faces and cell interiors.*

For a localized minimum (or maximum) the minimum (or maximum) is completely surrounded by larger (or smaller) values. In both cases, n_z is zero. If n_z is one, each neighborhood contains points with the same function value. Each of these points of same value would also be a minimum according to the original definition in [85]. (In that definition the zero set is also included in the positive and negative components.) At a saddle, surface components merge at a point or along a region. The zero set corresponds to the level set of the saddle value with the currently considered location "cut out." For a localized saddle, the level set at a saddle corresponds to several surface components that meet in a point. If this point is removed, these surface components become disconnected, *i.e.*, $n_z > 1$. If the saddle is along a region, the surface components meet along that region. If one location is cut out, these components remain connected, *i.e.*, $n_z = 1$. Thus, the number of components of the zero set determines whether there are non-ordinary points in each neighborhood. This fact holds for the 3D case, but it does not hold for the 2D case. In the 2D case, the zero set would be partitioned into more than one components when a saddle is "cut out," regardless whether it is a face saddle or not.

The concepts of an extended minimum, maximum, and saddle in combination with flat points can be used to define larger regions that are critical, *i.e.*, larger regions around which isosurface topology changes. Therefore, the definitions of distance and neighborhood are extended to regions as follows:

Definition 7 (Region Distance) *Let R be a connected region and $\|\mathbf{x}\|$ be a norm. The distance $\|\mathbf{x} - R\|$ of a point $\mathbf{p}$ to a region is the infimum of distances of $\mathbf{p}$ to all locations inside that region, i.e., $\|\mathbf{x} - R\| := \inf\{\|x - y\|, y \in R\}$.*

Definition 8 (Region Neighborhood) *Let R be a connected region. The ϵ-neighborhood $U_\epsilon(R)$ of R consists of all locations $U_\epsilon(R) := \{x, \|\mathbf{x} - R\| < \epsilon\}$, i.e., that have a distance less than ϵ from R.*

Definition 9 (Classification Region) *A connected region $R \subset \mathbb{R}^3$ in space is a* classification region *if the following statement holds: Each point $\mathbf{x} \in R$ is a flat point, an extended minimum, an extended maximum, or an extended saddle according to Definition 6, and R has maximal size, i.e., each point in an ϵ neighborhood around R is an ordinary point according to Definition 6.*

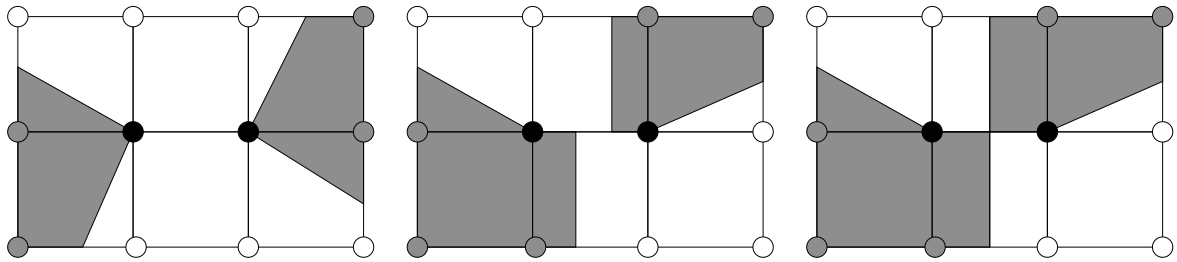


Figure 6.17: Different configurations for a constant edge: (a) The whole edge is a saddle. (b) Only a part of the edge is a saddle. (c) Only an isolated point on the edge is a saddle. Vertices of the classification regions are drawn as solid black disks. Vertices with an associated value larger than the value of the vertices of the classification regions are drawn as gray discs. Vertices with an associated value smaller than the vertices of the classification region are drawn as white discs.

Remark 7 *A Classification Region consists of points that cannot be classified locally but must be classified by also considering surrounding locations.*

Definition 10 (Regular and Critical Regions) *Let $M \subset \mathbb{R}^3$ be a mesh and $F : M \to \mathbb{R}$ be a C^0-continuous function that is C^∞-continuous in each grid cell. A classification region $R \subset \mathbb{R}^3$ is called (a) regular, (b) minimum, (c) maximum, or (d) saddle of F, if for all $\epsilon > 0$ there exists a neighborhood $U \subset U_\epsilon(R)$ with the following properties: If $\dot{\bigcup}_{i=1}^{n_p} P_i$ is a partition of the preimage of $(F(R), +\infty)$ in $U - R$ into "positive" connected components, $\dot{\bigcup}_{j=j}^{n_n} N_j$ is a partition of the preimage of $(-\infty, F(R))$ in $U - R$ into "negative" connected components, and $\dot{\bigcup}_{k=1}^{n_z} Z_k$ is the partition of the preimage of $\{F(R)\}$ in $U - R$ into zero set connected components, then (a) $n_p = n_n = n_z = 1$, (b) $n_p \geq 1$ and $n_n = n_z = 0$, (c) $n_n \geq 1$ and $n_p = n_z = 0$, and (d) $n_p + n_n > 2$ and $n_z = 1$.*

6.7 Representing Region Connectivity with Graphs

My method classifes regions consisting of grid vertices that have equal value and are connected via grid edges. This approach correctly detects extended minima and maxima since minima and maxima are always separated from other regions of equal value, *i.e.*, they are always surrounded by regions of a different value. Saddles, on the other hand, pose a problem. The neighborhood of a saddle always contains regular points that have the same associated value as the saddle. Figure 6.17 shows an example for this in two dimensions. Even though it is possible that the whole classified edge constitutes a saddle, see Figure 6.17(a), it is also possible that only a part of the edge, see Figure 6.17(b), or an isolated point on the edge, see Figure 6.17(c), constitutes a saddle. If one considers a saddle point coinciding with the edge of a cell, both boundary vertices of the edge have the same value. It is possible that my method classifies a region that is too large as saddle, or it may even merge several saddles, see Figure 6.18(a), and considers them as

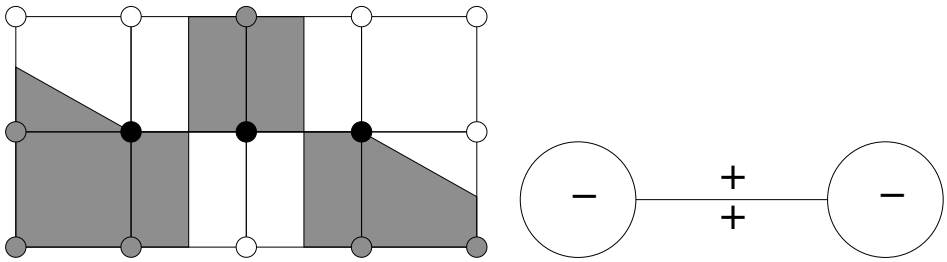


Figure 6.18: Limitations of my method: (a) When classifying a region consisting of edge connected vertices, it is possible that my method "merges" two (or more) distinct saddle points (or regions). (b) When a classified region has non-trivial topology, it is possible that a saddle is missed. When considering this image in 3-d by revolving it around the center, it consists of one positive and one negative region separated by a "torus" whith a disc in its middle. When classified the whole constant region is classified it is considered a regular region. When only the classification region consisting of extended minima, maxima, saddles and flat points, *i.e.*, the disc region at the center is classied, a saddle is detected.

one classification region. When a classification region has "non-trivial" topology, it is possible that a saddle is missed, see Figure 6.18(b). My approach detects saddles, when the "extended classification region," *i.e.*, the classification region and all edge connected "regular" vertices of the same value, does not contain a set of regular points that connects both negative and positive regions of a saddle. I suspect my algorithms fails, when a close non-manifold classification region occurs, *i.e.*, when a closed isosurface component for an isovalue equal to the saddle value coincides with grid vertices. I assume that the region has to be closed to connect both positive and negative regions of a saddle. Furthermore, as it corresponds to an isosurface component for the saddle value, it must be non-manifold. Though very unlikely, this situation can theoretically occur.

Lemma 6 *The union of a connected set S_R of regular points of equal value considered as whole is regular.*

Proof
In the context of my method, a connected region means a pathwise-connected region. Each point $\mathbf{x}$ in the set is a regular point. Consequently, its ϵ-neighborhood $U_\epsilon(\mathbf{x})$ contains exactly one positive and one negative (pathwise-)connected region. Without loss of generality, assume that the Euclidean norm is used, *i.e.*, the neighborhood is a sphere. It is necessary to show that the neighborhood around S_R consists of exactly two pathwise-connected regions. This is shown for the positive region. The same holds for the negative region by analogy. Around each point in $\mathbf{x}$ exists an ϵ-sphere such that the isosurface for the value of S_R partitions $U_\epsilon(\mathbf{x})$ into three parts: the isosurface, a positive region, and a negative region. Considering the union of all positive parts of all $U_\epsilon(\mathbf{x})$ for all $\mathbf{x}$ in S_R it must be shown that they are path-connected, *i.e.*, that a path between two arbitrary positive in the union of all positive regions exists. Let $\mathbf{a}_1$ and $\mathbf{a}_2$ be these points.

By choice of these points they both lie in a neighborhood of two points $\mathbf{x}_1 \in S_R$ and $\mathbf{x}_2 \in S_R$. Since S_R is path-connected, there exists a path between $\mathbf{x}_1$ and $\mathbf{x}_2$. Since a path is the image of a continuous function of the compact interval $[0, 1]$, the path is compact. Consequently, the path lies within the union of a finite number of spheres U_i, *i.e.*, , the path lies completely within the sets $U_\epsilon(\mathbf{x}_1) = U_1, U_2, \ldots U_n = U_\epsilon(\mathbf{x}_2)$. The path between $\mathbf{a}_1$ and $\mathbf{a}_2$ is constructed as follows. A positive position $\mathbf{p}_i \in U_i \cap U_{i+1}$ is chosen. Since the positive parts of each neighborhood are pathwise-connected, there exists a path between $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ as they both lie within U_{i+1}. Furthermore, there exists a path between $\mathbf{a}_1$ and $\mathbf{p}_1$ as they both lie within U_1, and there exists a path between $\mathbf{a}_2$ and $\mathbf{p}_{n-1}$ as they both lie within U_n. Consequently, $\mathbf{a}_1$ and $\mathbf{a}_2$ are pathwise connected. □

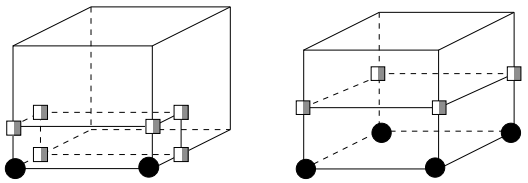
Regions are classified by constructing a graph whose connected components correspond to connected regions in a neighborhood around a region. This is done by considering all cells that have at least one vertex that belongs to the classification region. A graph whose nodes corresponds to edges of these cells and whose connected components correspond to connected positive and negative regions in space is constructed. Within a cell, one is concerned with one trilinear function. In the following, it is shown that when restricted to the trilinear interpolant in a cell, each connected region in the neighborhood contains at least one edge of that cell. Consequently, it is possible to represent these regions by cell edges.

Lemma 7 *Let C be a rectilinear cell with trilinear interpolation applied to it and v be an arbitrary value. Each maximum-extended connected positive region $R \subset f^{-1}\big((v, +\infty)\big)$ contains at least one vertex of C.*

Proof:
A maximum-extended connected region with values $> v$ contains a local maximum within its compact closure. Since trilinear interpolation only allows a maximum in the vertices of a cell to exist, R contains a vertex. □
(The same lemma holds for negative connected regions.)

Lemma 8 *Let C be a trilinearly interpolated rectilinear cell that intersects the classification region R, i.e., a cell that contains a number of vertices that belong to the classification region with a value of v. Each connected positive or negative region in the neighborhood of R intersecting C, i.e., $U_\epsilon(R) \cap C$, contains a part of an edge of the cell that starts at a vertex belonging to the classification region. (The vertex itself does not belong to the region.)*

Proof (sketch) Consider the L^∞-norm. A neighborhood around a point is a cube, and a neighborhood around an edge or a face is a (rectilinear) cuboid/box. The neighborhood around a collection of points, edges, and faces is a set of boxes. The intersection of the neighborhood with the cell is also a set of boxes, see Figure 6.19. Values are trilinearly interpolated within each box. Vertices of a box coincide with vertices of the cell, its edges, or lie in its interior. If a sufficiently small neighborhood is chosen, the vertices in the interior have the same polarity as one of their edge-connected neighbor vertices. Each connected region in one of the boxes is connected to one of the vertices of that box. This vertex, in turn, is connected to a vertex that lies either on an edge of C or coincides with one of its vertices belonging to the classification region. Thus,



(a) (b)

Figure 6.19: With respect to the L^∞-norm, the intersection of the neighborhood with a cell is a collection of boxes. If the neighborhood is chosen sufficiently small, vertices of the boxes that do not coincide with cell vertices or faces have the same polarity as an edge-connected vertex, and their corresponding region is also connected to one of the cell's edges.

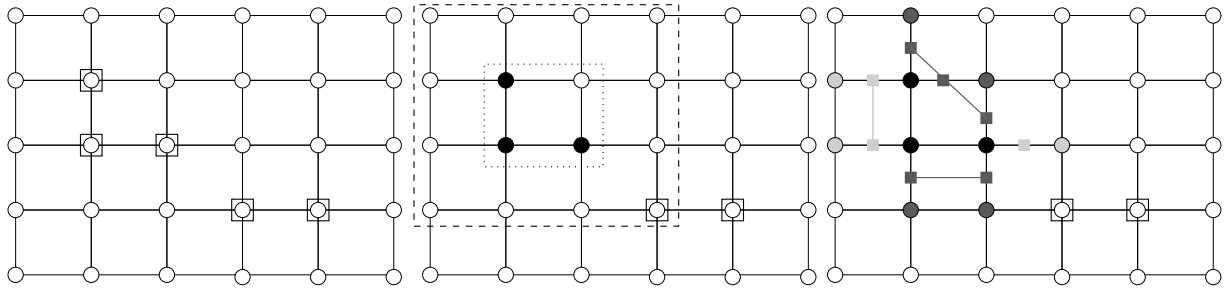
each positive- and each negative-connected component within the neighborhood is connected to an edge emanating from a vertex of the classification region. □
Furthermore, if two edges of same polarity are connected within the intersection of a cell's face with the neighborhood of the classification region, their corresponding regions are also connected. Additionally, "being connected" is a transitive property, *i.e.*, , if edge (region) A is connected to edge (region) B and edge (region) B is connected to edge (region) C, then edge (region) A and edge (region) C are connected as well.

6.8 Detecting Critical Regions

6.8.1 Overview

My algorithm detects critical regions in a two-pass approach. First, a pass is performed that classifies all grid vertices that can be classified locally, *i.e.*, all grid vertices whose edge-connected neighbors differ in value. This local classification is performed as described in Section 6.3. Using the polarities of the six edge-connected neighbor vertices, an index for a 64 entry LUT is computed. The type of vertex is then read from a manually generated LUT. Vertices that cannot be classified locally are marked by setting an associated flag, see Figure 6.20(a).

Second, this flag is used in a subsequent pass that handles global classification. The next vertex that belongs to a region that needs to be classified is located. Starting with this seed vertex a "flood fill" operation is performed that recursively adds all vertices having the same associated scalar value as the seed vertex and are edge-connected to the current vertex. While adding these vertices to the classification region, the associated flag that marks the vertex as belonging to an unclassified region is cleared. In addition, an updated bounding box that contains all marked vertices is maintained. After applying the flood-fill process this bounding box, shown as dotted rectangle in Figure 6.20(b), contains all vertices belonging to the current classification region. All vertices belonging to this region are flagged. The bounding box containing all vertices of the classification region is extended to a bounding box of grid cells such that no vertex belonging to the classification regions lies on the boundary of this bounding box, shown as dashed rectangle



(a) (b) (c)

Figure 6.20: Stages of the algorithm: (a) After the local pass all vertices that cannot be classified locally are marked with a flag (indicated by a rectangle around these vertices in the figure). Once the first flagged vertex is determined a flood fill is performed that marks all vertices belonging to the classification section (solid black disks in the figure). The region is classified by constructing two graphs, one representing positive- and one representing negative-connected regions in the neighborhood around the classification region. For each edge originating in a vertex belonging to the classification region, a corresponding node in the graph exists (shown as dark and light gray solid rectangles). Two nodes in the graph are connected when the corresponding edges belong to the same connected region in a cell. A region can be classified by counting the connected components in the two graphs.

in Figure 6.20(b).

A region is classified by constructing two graphs whose connected components correspond to connected regions in a neighborhood around that region. One graph corresponds to positive regions, and another graph corresponds to negative regions. These graphs are constructed by traversing all cells of the extended bounding box and adding nodes and edges to this graph cell-by-cell. It follows from the observations in Section 6.7 that for each cell a connected region in the neighborhood of the classification region contains at least one edge starting from a vertex of the classification region. Thus, connectivity of these regions can be represented by considering all edges that originate in a vertex belonging to the classification region. For each of those edges, a node in one of the graphs representing the classification region neighborhood is created. To add edges as nodes to one of the graphs, a unique identification number is assigned to each edge in a grid by effectively numbering all edges of the grid. This step ensures that the same node in the neighborhood graph is accessed for two cells sharing that edge.

Once the two graphs representing the neighborhood of a region are constructed, a region can be classified by counting the connected components in them. The graph in Figure 6.20(c), for example, indicates that the classified region is a saddle. When a region is classified, the search for the next vertex belonging to an unclassified region proceeds until all vertices of the grid are checked.

6.8.2 Constructing the Connectivity Graphs within a Cell

Within a cell the graph is constructed based on the vertex configuration within the cell. A main case is determined based on which vertices of the current cell belong to the classification region. The vertices of a cell are numbered according to Figure 2.22, and a look-up index is computed by setting the corresponding bit for each vertex belonging to the classification region. This step results in an index referencing one of 256 possible cases. Considering rotational symmetry, this number can be reduced to 23 base cases, shown in Figure 6.21. (This approach is similar to symmetry considerations for the MC method, and my numbering of base cases corresponds to the numbering scheme used in [67].)

An LUT that contains the base case number and permutations of vertices, edges, and faces is referenced that map vertex, edge, and face numbers of the current case to vertex, edge, and face numbers of the corresponding base case. Subsequently, the edge graph for a cell is constructed by calling a function that handles the corresponding base case. Whenever this function references a vertex, edge, or face of the cell it does so via the corresponding permutation function for the current topology case index.

All edges connecting the classification region to positive vertices are added as nodes to the positive-components graph and all edges connecting the classification region to negative vertices to the negative-components graph. If a pair of edges belongs to the same connected region in a cell, their corresponding nodes in the positive or negative edge graph are connected. When cell marching terminates, the connected components in one graph correspond to positive-connected regions, and the connected components in the other graph correspond to negative-connected regions.

First, consider cases $C0$ and $C22$. For case $C0$, a cell does not contain any vertices of the classification region and thus does not intersect the ϵ-neighborhood of the classification region. A $C22$ cell belongs completely to the classification region and does not intersect the neighborhood either. Cells of these types can be skipped during graph construction.

Cases $C3$, $C4$, $C6$, $C7$, $C10$, and $C13$ are handled by considering them as a combination of single vertices or single edges belonging to the classification area. If one considers, for example, case $C3$, one observes that two vertices belong to the classification region. Both vertices have edge-connected neighbor vertices that do not belong to the classification region. Thus, it is possible to handle them independently and construct the neighborhood graphs for each vertex individually. Similarly, case $C6$, for example, can be handled by constructing the graphs for an edge and for a vertex.

Constructing the neighborhood graph for a single vertex corresponds to case $C1$. This is equivalent to the situation in a single cell as described in Section 6.3. The neighborhood around the vertex in the classification region is partitioned in the same way as in a linearly interpolated tetrahedron. Thus, in this case all vertices of the same polarity are always connected, and it is only necessary to add edges between all edges of equal polarities in the corresponding graph.

For the construction of the neighborhood graph of an edge, it is necessary to consider 16 cases. (The edge is connected to four vertices that do not belong to the classification region.) By using symmetry and reversing polarities, it is possible to reduce the number of cases to four possible sub-cases, see Figure 6.22. Figure 6.22 shows possible configurations for the edge-

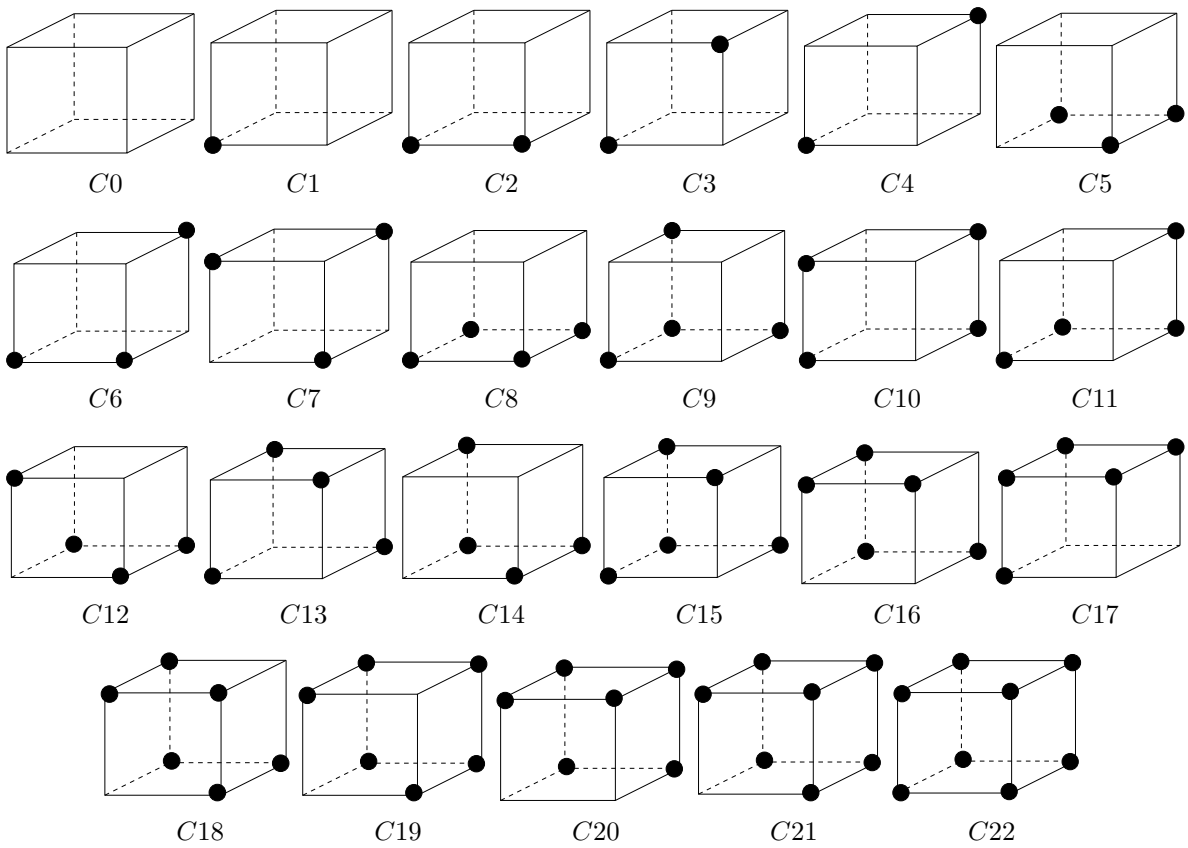


Figure 6.21: Base cases for constructing graphs that represent positive and negative regions around a classification region. A case number is determined according to which vertices belong to the classification region (marked as solid black disks). The other vertices can assume arbitrary values different from the value of the classification region.

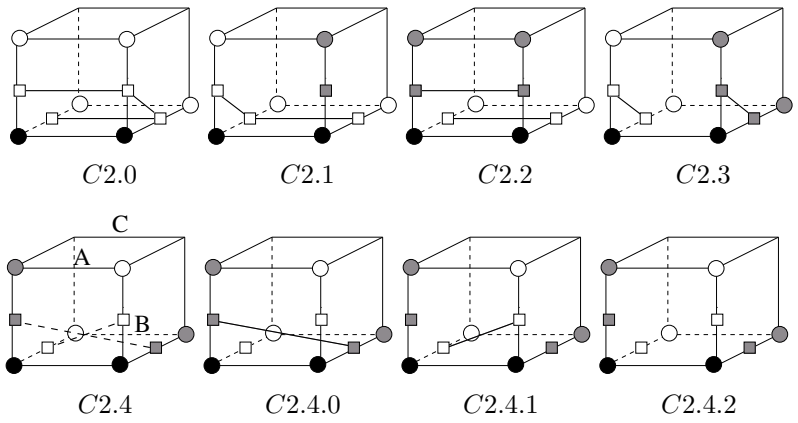


Figure 6.22: Sub-cases for connecting an edge.

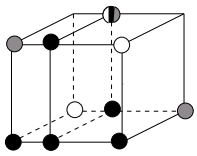


Figure 6.23: Disambiguation for case $C2.4$.

connected vertices that have either positive (gray) or negative (white) polarity. Each edge has a corresponding node in the classification graph (shown as white rectangles for the negative graph and gray rectangles for the positive graph). Two nodes are connected by a line segment when they must be connected in their graph.

Cases $C2.0$ to $C2.3$ are derived from this observation: Nodes of the same polarity are on a face, and they are connected on the intersection of that face with the neighborhood of the classification region. Since connectivity is transitive, the actual connectivity graph is the transitive hull of the graphs shown in the figure. However, only the number of connected components in a graph is of interest. Consequently, it suffices to add those edges to the graphs that are shown in the figure. For case $C2.4$ two diagonally opposite vertices have the same polarity. In this case, it is necessary determine whether the positive vertices are connected ($C2.4.0$), whether the negative vertices are connected ($C2.4.1$), or whether all regions are separated ($C2.4.2$). To distinguish between these cases, it is determined, for which parameter values t_A and t_B along the edges "A" and "B" the value of the classification region is taken on. (For both parameter values, zero corresponds to the "left" end of the edge and one to the "right" end of the edge.) If $t_A < t_B$ then the regions belonging to v_2 and v_4, *i.e.*, the negative regions, overlap. If $t_A > t_B$, the regions belonging to vertices v_3 and v_5, *i.e.*, the positive regions, overlap. If $t_A = t_B$, the bilinear slice through the cell at that parameter value contains three vertices with the value of the classification region. The remaining vertex on edge "C" determines the polarity of the whole face. If it is positive, the positive vertices v_3 and v_5, along with their corresponding edges, are connected ($C2.4.0$); if it is negative, the negative vertices v_2 and v_4, along with their corresponding edges, are connected ($C2.4.1$). If it also has the value of the classification region, all vertices and edges are separated ($C2.4.2$). The remaining sub-cases can be derived from these base cases.

For the remaining base cases, those vertices that do not belong to the classification region, but are connected to one of its vertices by an edge, are numbered. Subsequently, a topology sub-case number is computed by setting the bit for each positive vertex. These sub-cases can be derived from a number of base sub-cases. Figure 6.24 shows base sub-configurations for topology base case $C5$. (Edges e_8 and e_0 lead to the same non-classification region vertex on a face and are connected within the intersection of a neighborhood of the classification region with that face. Thus, their corresponding nodes in the classification graph are always connected.)

Similarly to the construction of the classification graph for an edge, the graphs for most cases can be derived from the fact that nodes for edges on the same face are connected within the intersection of the region neighborhood and that face if they have the same polarity and using transitivity of the graph. Similarly to the edge case $C2$, only a minimum number of edges needed to obtain a correct number of connected components is given for the neighborhood graphs. The ac-

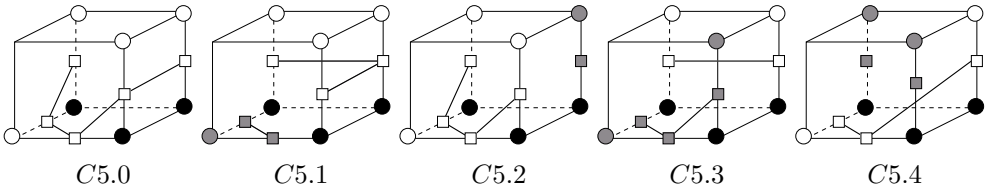


Figure 6.24: Sub-cases for topology base case $C5$.

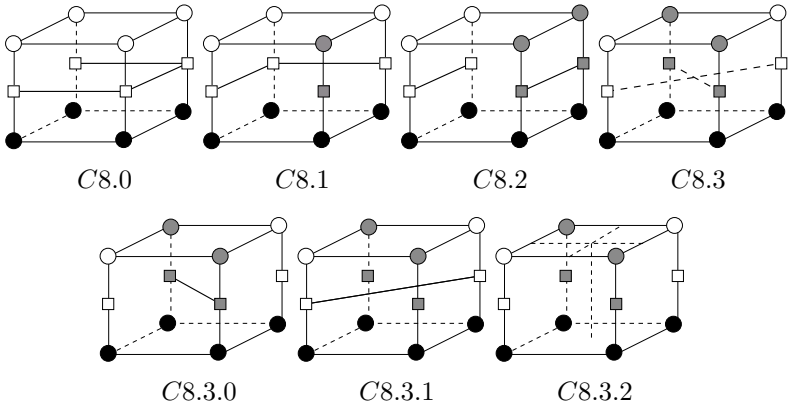


Figure 6.25: Sub-cases for topology base case $C8$.

tual neighborhood graphs are the transitive closure of the graphs in the figure. Sub-configuration $C5.4$ is interesting: In contrast to sub-configuration $C2.4$ no ambiguity exists. The bottom face has always the same polarity as vertex v_0. Since trilinear interpolation is continuous, a bilinear slice close to the bottom face connects the vertices of the same polarity as v_0, *i.e.*, v_0 and v_6. Thus, nodes for edges leading from the classification region to these vertices must be connected in the classification graph.

Case $C8$, see Figure 6.25, is similar to the case of handling an edge belonging to the classification region. For cases $C8.0 - 8.2$ the same arguments can be used to derive the classification graphs in each cell. It differs from the edge case in the way ambiguities ($C8.3$) are resolved: All vertices of the bottom face have the same value. Thus, contours on all bilinear slices parallel to the top face are topologically equivalent to contours on the top face. The asymptotic decider [68] is used to determine connectivity on that face and connect negative nodes, or positive nodes. If the saddle on the top face has the same value as the classification region, each bilinear slice parallel to that face has the same saddle value. In this case, the saddle on that face is connected to the classification region via an "extended" internal cell saddle (see Section 6.9.1). To classify the region correctly, it must be determined, which regions are connected beyond that face, see Section 6.9.1.

The neighborhood graphs for base case $C9$, see Figure 6.26, can be constructed using the same arguments as for case $C5$.

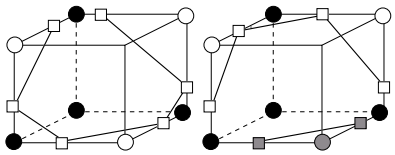


Figure 6.26: Sub-cases for topology base case $C9$.

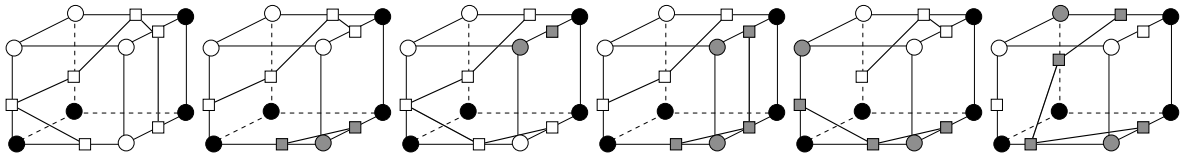


Figure 6.27: Sub-cases for topology base case $C11$.

Figure 6.27 shows the classification graphs for sub-cases of topology case $C11$. The graphs for cases $C11.0$-$C11.4$ can be constructed using the same observations as for base case $C5$. Observe, that for sub-case $C11.5$ all positive regions are connected since the complete bottom and back faces have positive polarity.

Case $C12$ is a combination of base case $C5$ and vertex case $C1$. Case $C14$ can be obtained by "mirroring" base case $C11$. Base case $C15$ is a combination of base case $C9$ and vertex case $C1$. Base case $C16$ has only three vertices that do not belong to the classification region. If a pair of vertices has the same polarity, it is alway connected in the neighborhood of the classification region. For base cases $C17$, $C19$, and $C20$, all vertices not belonging to the classification region lie on a single face. Furthermore, on this face they are always connected, when they have the same polarity. Thus, in these cases vertices of equal polarity are always connected. Base case $C18$ contains only two vertices that do not belong to the classification region. If these vertices have the same polarity, they are connected. Case $C21$ consists of only one vertex. In this case, it is necessary to connect the nodes in the classification graph that correspond to its incoming edges.

6.8.3 Computing Connected Components in the Connectivity Graph

To compute the number of connected components in the neighborhood graph, a variant of the *union-find* algorithm is used see [44, 75]. Since only connected components of the graph are of interest, connectivity information between notes is not explicitly stored. Instead, the graph is treated as as a set of connected components. Each connected component is a set of node identifiers (ids) of all nodes belonging to that connected component. As a new nodes is added to the graph, a new connected component, *i.e.*, a set with a the node index as single element, is added to the list of connected components. If an edge between two nodes belonging to different

connected components is added, the union of these sets is computed and replaces the two sets corresponding to the individual connected components.

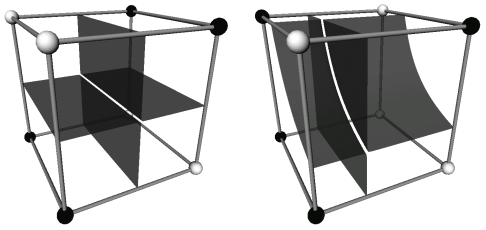
Those sets are managed with the union-find algorithm [44, 75]. The sets are stored as a *forest of trees*, *i.e.*, each set is stored as a tree consisting of all elements of the set. Each set is represented by the root of its storage tree. To determine whether two elements belong to the same set, the root for the tree containing each element is determined. If both roots are the same, the elements belong to the same set. If they are different, they belong to different sets. Since it is only necessary to traverse trees toward their root node, individual trees are stored in *parent* representation, *i.e.*, each tree node contains a data entry listing the element and a pointer to the parent of that node. The root contains a null-pointer indicating that it does not have a parent. To find the representing element of a set, *i.e.*, the root node of the tree corresponding to that set, for a given element, the parent pointers are followed, until the root is found. Two sets are joined in a union, by replacing the null-pointer in the representer of one set with a pointer to the representer of the other set. Two optimizations of the union-find algorithm, described by Sedgewick [75], are used: *path-length compression* reduces the height of a tree. Whenever the representing node for a set element is determined, the path toward the root is followed a second time replacing the parent pointer with a direct pointer to the root of the tree representing the set. Furthermore, a *weight* is maintained for each root-node that counts the number of elements in the tree below it. Whenever join two trees are joined, choose the tree with the larger number of elements is chosen as the new root instead of making an arbitrary choice.

Instead of counting the number of components (or sets) after generating the graph, the number of currently existing sets is stored and updated. This number is initialized as zero. Whenever a new set/connected component is added, this number is increased by one. When two independent sets/connected components are joined, it is decreased by one. A hash-table is maintained to locate the corresponding element entry for a given node identifier efficiently.

6.9 Extended Interior Saddles and Face Saddles

6.9.1 Introduction

Section 6.3 stated that a saddle of the bilinear interpolant on a boundary face of a cell may or may not be a saddle of piecewise trilinear interpolation. By determining whether the positive or negative regions of the boundary face saddle are connected within both adjacent cells, it was possible classify a face saddle as saddle or regular point of piecewise trilinear interpolation. Even if vertex values along an edge are unique, it is possible that neither positive or negative connections are connected within a cell, see, for example Figure 6.7. Section 6.3 discarded these “extended” saddles as it was only concerned with isolated critical points. When detecting critical regions, one must reconsider extended saddles. Furthermore, if the requirement that vertices connected by an edge differ in value is discarded, additional extended saddles in a cells interior can occur, see Figure 6.28. In these cases, a cell has more than one face containing a saddle of equal value. When examining the gradient ∇T of the trilinear interpolant within a cell, it turns out that it vanishes along a straight line, see Figure 6.28(a) or a hyperbola on a face, see Figure 6.28(b).



(a) (b)

Figure 6.28: Extended interior saddle of a cell connecting two faces with saddles having the same associated value. (a) Opposite faces are saddle-connected. (b) Two neighboring faces are saddle-connected.

Each location on this curve is a saddle of a bilinear slice through the cell. This extended interior saddle "connects" two boundary faces with equal saddle value. Consequently, it is not possible to treat face saddles and interior saddles independently. (Only internal saddles that are isolated points can still be classified individually.) Whenever the criterion from Lemma 5 fails to determine whether positive or negative regions of the saddle on the boundary face are connected, an extended interior saddle exists, and neither positive nor negative regions of the face saddle are connected within that cell. It is not possible to determine, locally, whether the face saddle is a saddle of piecewise trilinear interpolation. Instead, this saddle must be "traced" through cells, constructing graphs that represent positive and negative regions in its neighborhood. This is analogous to the scheme used to classify regions consisting of vertices of constant value.

The remainder of this section proceeds as follows. First, the solutions to $\nabla T = 0$ that do not correspond to isolated critical points are examined. Subsequently, it is shown that if the criterion from Lemma 5 fails to determine whether positive or negative regions are connected in a cell, in fact neither regions are connected. Instead, the gradient of the trilinear interpolant vanishes along a curve connecting the saddle on that face with one or more saddles of equal value on other faces of the cell. Subsequently, this analysis is used to derive a scheme that allows the classification of face saddles by tracing an extended saddle region by following extended internal saddles and constructing two graph representing positive and negative connected components in a neighborhood around that saddle.

6.9.2 Extended Interior Saddles

Lets reconsider interior saddles. Within a cell, trilinear interpolation is a C^∞–continuous function and can be written in the form

$$T(x, y, z) = axyz + bxy + cyz + dxz + ex + fy + gz + h \;, \tag{6.6}$$

assuming that the domain is the unit cube $[0,1]^3$. The coefficients can be calculated from the vertex values as

$$\begin{aligned} a =& v_1 + v_3 + v_4 + v_6 - v_0 - v_7 - v_5 - v_2 \ , \\ b =& v_0 - v_1 + v_2 - v_3 \ , \\ c =& v_0 - v_3 - v_4 + v_7 \ , \\ d =& v_0 - v_1 - v_4 + v_5 \ , \\ e =& v_1 - v_0 \ , \\ f =& v_3 - v_0 \ , \\ g =& v_4 - v_0 \text{ , and} \\ h =& v_0 \ . \end{aligned} \tag{6.7}$$

The vertex values can be computed from the coefficients as

$$\begin{aligned} v_0 =& h \ , \\ v_1 =& e + h \ , \\ v_2 =& b + e + f + h \ , \\ v_3 =& f + h \ , \\ v_4 =& g + h \ , \\ v_5 =& d + e + g + h \ , \\ v_6 =& a + b + c + d + e + f + g + h \text{ , and} \\ v_7 =& c + f + g + h \ . \end{aligned} \tag{6.8}$$

Critical points occur at locations where the gradient vanishes, see Section 2.1.2. Using this criterion on Equation 6.6 obtains

$$\left.\begin{aligned} \frac{\partial T}{\partial x} &= ayz + by + dz + e = 0 \\ \frac{\partial T}{\partial y} &= axz + bx + cz + f = 0 \\ \frac{\partial T}{\partial z} &= axy + cy + dx + g = 0 \end{aligned}\right\} \equiv \nabla T(x,y,z) = 0 \tag{6.9}$$

Desired are solutions of this equation. The Hessian of T is

$$H(x,y,z) = \begin{pmatrix} 0 & az+b & ay+d \\ az+b & 0 & ax+c \\ ay+d & ax+c & 0 \end{pmatrix} \ . \tag{6.10}$$

Eigenvalues λ_i of the Hessian are the solutions to the following equation

$$-\lambda^3 + k_x^2\lambda^2 + k_y\lambda^2 + k_z\lambda^2 + 2k_xk_yk_z = 0 \ , \tag{6.11}$$

where

$$k_x = ax + c \,, \tag{6.12}$$
$$k_y = ay + d \text{ , and} \tag{6.13}$$
$$k_z = az + b \,. \tag{6.14}$$

Since $H(x, y, z)$ is symmetric, it has three real-valued eigenvalues. The determinant of the Hessian equals to the product of the eigenvalues, *i.e.*,

$$\begin{vmatrix} 0 & az+b & ay+d \\ az+b & 0 & ax+c \\ ay+d & ax+c & 0 \end{vmatrix} = 2(ax+c)(ay+d)(az+b) = \lambda_1\lambda_2\lambda_3 \,. \tag{6.15}$$

As a result of Lemma 1, all points for which the determinant of the Hessian differs from zero are saddles.

If a z-axis perpendicular slice is considered, trilinear interpolation reduces to bilinear interpolation on that slice, *i.e.*, interpolated values are obtained as

$$bi'_z(x, y) = v'_0(z)(1-x)(1-y) + v'_1(z)x(1-y) + v'_2(z)xy + v'_3(z)(1-x)y \,, \tag{6.16}$$

where

$$\begin{aligned} v'_0(z) =& (1-z)v_0 + zv_4 = gz + h \,, \\ v'_1(z) =& (1-z)v_1 + zv_5 = dz + gz + e + h \,, \\ v'_2(z) =& (1-z)v_2 + zv_6 = az + cz + dz + gz + b + e + f + h \text{ , and} \\ v'_3(z) =& (1-z)v_3 + zv_7 = cz + gz + f + h \,. \end{aligned} \tag{6.17}$$

This equation can be written as

$$bi'_z(x, y) = a'(z)xy + b'(z)x + c'(z)y + d'(z) \,, \tag{6.18}$$

with

$$\begin{aligned} a'(z) =& \; v'_0(z) - v'_1(z) + v'_2(z) - v'_3(z) &&= az + b \,, \\ b'(z) =& \; v'_1(z) - v'_0(z) &&= dz + e \,, \\ c'(z) =& \; v'_3(z) - v'_0(z) &&= cz + f \text{ , and} \\ d'(z) =& \; v'_0(z) &&= gz + h \,. \end{aligned} \tag{6.19}$$

The derivatives of the bilinear interpolant are

$$\begin{aligned} \frac{\partial bi'_z}{\partial x} &= a'(z)y + b'(z) = ayz + by + dz + e = \frac{\partial F}{\partial x} \text{ , and} \\ \frac{\partial bi'_z}{\partial y} &= a'(z)x + c'(z) = axz + bx + cz + f = \frac{\partial F}{\partial y} \,. \end{aligned} \tag{6.20}$$

The first derivatives on the bilinear slice are equal to the first derivatives of the trilinear interpolant in that direction. By symmetry this property also holds for y- and z-axis perpendicular slices. The eigenvalue of the Hessian of the bilinear interpolants are according to Equation 2.39

$$\lambda_{z\,1/2} = \pm a'(z) = \pm(az + b) = \pm k_z \,. \tag{6.21}$$

Analogously, one obtains

$$\lambda_{x\,1/2} = \pm k_x \text{ , and} \tag{6.22}$$
$$\lambda_{y\,1/2} = \pm k_y \text{ ,} \tag{6.23}$$

as eigenvalues for the Hessian of the bilinear interpolant on x- and y-axis perpendicular slices.
Case 1: $a \neq 0$
Equation 6.9 can be rewritten as

$$\frac{\partial T}{\partial x} = ayz + by + dz + e = \underbrace{a\left(y + \frac{d}{a}\right)\left(z + \frac{b}{a}\right)}_{ayz+by+dz+\frac{bd}{a}} + \underbrace{\overbrace{\frac{ae - bd}{a}}^{=:a_x}}_{e-\frac{bd}{a}} = 0 \tag{6.24}$$

$$\frac{\partial T}{\partial y} = axz + bx + cz + f = a\left(x + \frac{c}{a}\right)\left(z + \frac{b}{a}\right) + \overbrace{\frac{af - bc}{a}}^{=:a_y} = 0 \tag{6.25}$$

$$\frac{\partial T}{\partial z} = axy + cy + dx + g = a\left(x + \frac{c}{a}\right)\left(y + \frac{d}{a}\right) + \overbrace{\frac{ag - cd}{a}}^{=:a_z} = 0 \tag{6.26}$$

Case 1.1: $a_x \neq 0$; $a_y \neq 0$; $a_z \neq 0$
If follows that $x \neq -\frac{c}{a}$, $y \neq -\frac{d}{a}$ and $z \neq -\frac{b}{a}$, otherwise there is a contradiction.
From Equation 6.25 it follows that

$$z + \frac{b}{a} = -\frac{a_y}{a^2\left(x + \frac{c}{a}\right)} \text{ .} \tag{6.27}$$

From Equation 6.26 it follows that

$$y + \frac{d}{a} = -\frac{a_z}{a^2\left(x + \frac{c}{a}\right)} \text{ .} \tag{6.28}$$

Using Equations 6.27 and 6.28 in Equation 6.24 yields

$$a\frac{-a_y}{a^2\left(x + \frac{c}{a}\right)} \cdot \frac{-a_z}{a^2\left(x + \frac{c}{a}\right)} + \frac{a_x}{a} = 0 \text{ .}$$

This equation can also be written as

$$\left(x + \frac{c}{a}\right)^2 = -\frac{a_y a_z}{a^2 a_x} \text{ .} \tag{6.29}$$

Analogously, one obtains

$$\left(y + \frac{d}{a}\right)^2 = -\frac{a_x a_z}{a^2 a_y} \text{ , and} \tag{6.30}$$
$$\left(z + \frac{b}{a}\right)^2 = -\frac{a_x a_z}{a^2 a_y} \text{ .} \tag{6.31}$$

Case 1.1.1: $a_x a_y a_z > 0$
$\Rightarrow$ No solution exists.

Case 1.1.2: $a_x a_y a_z < 0$
From Equation 6.29 it follows that

$$x_{1/2} = -\frac{c}{a} \pm \frac{\sqrt{-a_x a_y a_z}}{a a_x} \; . \tag{6.32}$$

Using Equation 6.32 in Equations 6.26 and 6.25 yields

$$y_{1/2} = -\frac{d}{a} \pm \frac{\sqrt{-a_x a_y a_z}}{a a_y} \text{ , and} \tag{6.33}$$

$$z_{1/2} = -\frac{b}{a} \pm \frac{\sqrt{-a_x a_y a_z}}{a a_z} \; . \tag{6.34}$$

Since $x \neq -\frac{c}{a}$, $y \neq -\frac{d}{a}$ and $z \neq -\frac{b}{a}$ holds, the determinant of the Hessian, see Equation 6.15, differs from zero and both solutions are saddles of the trilinear interpolant. Both saddles are not necessarily within the cells boundaries. $\Rightarrow$ There are up to two isolated saddles of the trilinear interpolant within the cell.

Case 1.2: $a_x = 0; a_y \neq 0; a_z \neq 0$
From $a_x = 0$ it follows by Equation 6.24 that

$$a \left(y + \frac{d}{a} \right) \left(z + \frac{b}{a} \right) = 0 \tag{6.35}$$

holds. From this equation it follows that either $y = -\frac{d}{a}$ or $z = -\frac{b}{a}$ (or both) holds. From $y = -\frac{d}{a}$ it follows by Equation 6.26 that $\frac{a_z}{a} = 0$ which leads to a contradiction. From $z = -\frac{b}{a}$ it follows by Equation 6.25 that $\frac{a_y}{a} = 0$ which also leads to a contradiction. Consequently, no solution exists for this case.

Case 1.3: $a_x \neq 0; a_y = 0; a_z \neq 0$
From $a_x = 0$ it follows by Equation 6.25 that

$$a \left(x + \frac{c}{a} \right) \left(z + \frac{b}{a} \right) = 0 \tag{6.36}$$

holds. From this equation it follows that either $x = -\frac{c}{a}$ or $z = -\frac{b}{a}$ (or both) holds. From $x = -\frac{c}{a}$ it follows by Equation 6.26 that $\frac{a_z}{a} = 0$ which leads to a contradiction. From $z = -\frac{b}{a}$ it follows by Equation 6.24 that $\frac{a_y}{a} = 0$ which also leads to a contradiction. Consequently, no solution exists for this case.

Case 1.4: $a_x \neq 0; a_y \neq 0; a_z = 0$
From $a_z = 0$ it follows by Equation 6.26 that

$$a \left(x + \frac{c}{a} \right) \left(y + \frac{d}{a} \right) = 0 \tag{6.37}$$

holds. From this equation it follows that either $x = -\frac{c}{a}$ or $y = -\frac{d}{a}$ (or both) holds. From $x = -\frac{c}{a}$ it follows by Equation 6.25 that $\frac{a_y}{a} = 0$ which leads to a contradiction. From $y = -\frac{d}{a}$ it follows by Equation 6.24 that $\frac{a_x}{a} = 0$ which also leads to a contradiction. Consequently, no solution exists for this case.

Case 1.5: $a_x = 0; a_y = 0; a_z \neq 0$
From $a_x = 0$ it follows according to Equation 6.24 that either $y = -\frac{d}{a}$ or $z = -\frac{b}{a}$ (or both) holds. Since $y = -\frac{d}{a}$ implies that $\frac{a_z}{a} = 0$, which leads to a contradiction, $z = -\frac{b}{a}$ must hold. (The case $z = -\frac{b}{a}$ implies that $\frac{a_y}{a} = 0$ which is true.) All solutions lie on the z-axis perpendicular plane $z = -\frac{b}{a}$. Further, Equation 6.26 must hold for x and y yielding the relationship

$$C(x, y) = axy + dx + cy + g = 0 \tag{6.38}$$

The curve consisting of all positions where Equation 6.38 holds is a bilinear contour. It is important to note, that the degenerate case, *i.e.*, all positions lie on a pair of axis-aligned asymptotes, cannot occur. The partial derivatives for $C(x, y)$ are

$$\begin{aligned} \frac{\partial C}{\partial x} &= ay + d \text{ , and} \\ \frac{\partial C}{\partial y} &= ax + c \text{ .} \end{aligned} \tag{6.39}$$

Thus, $C(x, y)$ has a critical point at $(-\frac{c}{a}, -\frac{d}{a})$, which is a saddle, see Section 2.2.5. The corresponding critical value is

$$a\frac{cd}{a^2} - \frac{cd}{a} - \frac{cd}{a} + g = g - \frac{dc}{a} = \frac{ag - cd}{a} = \frac{a_z}{a} \neq 0 \text{ .}$$

Consequently, the saddle value differs from zero, and the degenerate case does not occur for $C(x, y) = 0$.

Classical Morse theory does not apply for locations along this curve. The criticality is not localized and degenerate. Only two of the three eigenvalues of the Hessian differ from zero. However, if a bilinear slice through the cell at an arbitrary position of the curve is considered, the intersection point of the curve with that slice is a saddle on that slice. If one considers the slice $z = -\frac{b}{a}$ of the trilinear interpolant that contains the criticality, one obtain the following:

$$T\left(x, y, -\frac{b}{a}\right) = -\frac{bc}{a}y - \frac{bd}{a}x + ex + fy - \frac{gb}{a} + h \text{ .} \tag{6.40}$$

From $a_x = 0$ and $a_y = 0$ follows $bd = ae$ and $bc = af$, respectively. Using this relationship in Equation 6.40 yields

$$T\left(x, y, -\frac{b}{a}\right) = -\frac{af}{a}y - \frac{ae}{a}x + ex + fy - \frac{gb}{a} + h = h - \frac{gb}{a} \text{ .} \tag{6.41}$$

All positions within the slice, and, consequently, all saddles along the critical curve have the same value. According to Definition 6, consists of extended saddle points. Whether the whole

curve is a saddle, depends on which regions are connected in adjacent cell. (This is analogous to saddles on boundary faces.)

Case 1.6: $a_x = 0; a_y \neq 0; a_z = 0$
From $a_x = 0$ it follows according to Equation 6.24 that either $y = -\frac{d}{a}$ or $z = -\frac{b}{a}$ (or both) holds. Since $z = -\frac{b}{a}$ implies that $\frac{a_y}{a} = 0$ which leads to a contradiction, $y = -\frac{d}{a}$ must hold. (The case $y = -\frac{d}{a}$ implies that $\frac{a_z}{a} = 0$ which is true.) All solutions lie on the y-axis perpendicular plane $y = -\frac{b}{a}$. Further, Equation 6.25 must hold for x and z yielding the relationship

$$axz + bx + cz + f = 0 \; . \tag{6.42}$$

This case is analogous to Case 1.5.

Case 1.7: $a_x \neq 0; a_y = 0; a_z = 0$
From $a_y = 0$ it follows according to Equation 6.25 that either $x = -\frac{c}{a}$ or $z = -\frac{b}{a}$ (or both) holds. Since $z = -\frac{b}{a}$ implies that $\frac{a_x}{a} = 0$ which leads to a contradiction, $x = -\frac{c}{a}$ must hold. (The case $x = -\frac{c}{a}$ implies that $\frac{a_z}{a} = 0$ which is true.) All solutions lie on the x-axis perpendicular plane $x = -\frac{c}{a}$. Further, Equation 6.26 must hold for y and z yielding the relationship

$$ayz + by + dz + e = 0 \; . \tag{6.43}$$

This case is analogous to Case 1.5.

Case 1.8: $a_x = 0; a_y = 0; a_z = 0$
From Equations 6.24 to 6.26 follows that

$$\left(y = -\frac{d}{a} \vee z = -\frac{b}{a}\right) \wedge \left(x = -\frac{c}{a} \vee z = -\frac{b}{a}\right) \wedge \left(x = -\frac{c}{a} \vee y = -\frac{d}{a}\right) \; ,$$

where "$\vee$" indicates a logical or, and "$\wedge$" indicates a logical and. This statement is equivalent to

$$\left(x = -\frac{c}{a} \wedge y = -\frac{d}{a}\right) \vee \left(x = -\frac{c}{a} \wedge z = -\frac{b}{a}\right) \vee \left(y = -\frac{d}{a} \wedge z = -\frac{b}{a}\right) \; .$$

All potential critical points lie on three axis-aligned lines, *i.e.*,

$$x = -\frac{c}{a} \; ; \qquad y = -\frac{d}{a} \; ; \qquad z = t \in [0,1] \; , \tag{6.44}$$

$$x = -\frac{c}{a} \; ; \qquad y = t \in [0,1] \; ; \qquad z = -\frac{b}{a} \text{ , and} \tag{6.45}$$

$$x = t \in [0,1] \; ; \qquad y = -\frac{d}{a} \; ; \qquad z = -\frac{b}{a} \; . \tag{6.46}$$

If one considers slices perpendicular to one of these three lines, the bilinear interpolant on that slice has a saddle at the intersection point with that plane provided it does not contain the other

two lines, as $k_i \neq 0$ along an i-axis aligned line. At the intersection point if the three axis-aligned lines, all eigenvalues of the Hessian are zero. As shown in Case 1.5, it follows from $a_x = 0$ and $a_y = 0$ that the $z = -\frac{b}{a}$ slice has constant value. Analogously, it follows from $a_x = 0$ and $a_z = 0$, that the $y = -\frac{d}{a}$ slice has constant value, and from $a_y = 0$ and $a_z = 0$ follows that the $x = -\frac{c}{a}$ slice has constant value. All saddles of the bilinear slices along the three axis-aligned lines have constant value. According to Definition 6 each point on these lines is a saddle (with the exception of the intersection point which is a flat point). Whether the whole "saddle region" is a saddle depends on how the regions are connected outside the cell.

Case 2: $a = 0$
Equation 6.9 reduces to

$$\frac{\partial T}{\partial x} = by + dz + e = 0 \ , \tag{6.47}$$

$$\frac{\partial T}{\partial y} = bx + cz + f = 0 \text{ , and} \tag{6.48}$$

$$\frac{\partial T}{\partial z} = dx + cy + g = 0 \ . \tag{6.49}$$

The Hessian reduces to

$$H(x, y, z) = \begin{pmatrix} 0 & b & d \\ b & 0 & c \\ d & c & 0 \end{pmatrix} \ , \tag{6.50}$$

implying that $k_x = c$, $k_y = d$, and $k_z = b$.

Case 2.1: $b \neq 0; c \neq 0; d \neq 0 \Rightarrow bcd \neq 0$
Multiplying Equations 6.47, 6.48, and 6.49 by c, d, and b, respectively, results in the equations

$$(6.47) \cdot c: \quad bcy + cdz = -ce \ , \tag{6.51}$$

$$(6.48) \cdot d: \quad bdx + cdz = -df \text{ , and} \tag{6.52}$$

$$(6.49) \cdot b: \quad bdx + bcy = -bg \ . \tag{6.53}$$

Adding two of the above equations and subtracting the remaining equation results in

$$(6.48) + (6.49) - (6.47): \quad 2bdx = ce - df - bg \quad \Rightarrow x = \frac{ce - bg - df}{2bd} \ , \tag{6.54}$$

$$(6.47) + (6.49) - (6.48): \quad 2bcy = df - bg - ce \quad \Rightarrow y = \frac{df - bg - ce}{2bd} \text{ , and} \tag{6.55}$$

$$(6.47) + (6.48) - (6.49): \quad 2cdz = bg - ce - df \quad \Rightarrow z = \frac{bg - ce - df}{2cd} \ . \tag{6.56}$$

The solution is an isolated saddle point since the determinant of the Hessian differs from zero.

Case 2.2: $b = 0; c \neq 0; d \neq 0$
From Equation 6.47 it follows that $dz + e = 0$ must hold implying that $z = -\frac{e}{d}$ holds. From

Equation 6.48 it follows that $cz+f=0$ must hold implying that $z=-\frac{f}{c}$ holds. If $-\frac{e}{d} \neq -\frac{f}{c}$ then no solution exists. Otherwise, all possible solutions lie on a straight line within the $z=-\frac{e}{d}=-\frac{f}{c}$ plane that satisfies the equation

$$dx+cy+g=0 \tag{6.57}$$

which is derived from Equation 6.49. Like in Case 1.5, this case represents an extended, degenerate criticality. If one considers x- or y-axis perpendicular slices through the cell, the bilinear interpolant has a saddle at the intersection with the straight line implied by Equation 6.57, since $k_x=c\neq 0$ and $k_y=d\neq 0$. From $z=-\frac{e}{d}=-\frac{f}{c}$ it follows that

$$T\left(x,y,-\frac{e}{d}\right)=-\frac{cf}{c}y-\frac{de}{d}x+ex+fy-\frac{fg}{c}+h=h-\frac{fg}{c}\ . \tag{6.58}$$

All values in the slice, and, consequently, all bilinear saddle values are constant. Each point along the line is an extended saddle by Definition 6.

Case 2.3: $b\neq 0; c=0; d\neq 0$
From Equation 6.48 it follows that $bx+f=0$ must hold implying that $x=-\frac{f}{b}$ holds. From Equation 6.49 it follows that $dx+g=0$ must hold implying that $x=-\frac{g}{d}$ holds. If $-\frac{f}{b}\neq -\frac{g}{d}$ then no solution exists. Otherwise, all possible solutions lie on a straight line within the $x=-\frac{f}{b}=-\frac{g}{d}$ plane that satisfies the equation

$$by+dz+e=0 \tag{6.59}$$

which is derived from Equation 6.47. Analogously to Case 2.2, all locations along the straight line implied by Equation 6.59 are saddles of the bilinear interpolant on y- and z-axis perpendicular slices.

Case 2.4: $b\neq 0; c\neq 0; d=0$
From Equation 6.47 it follows that $by+e=0$ must hold implying that $y=-\frac{e}{b}$ holds. From Equation 6.49 it follows that $cy+g=0$ must hold implying that $y=-\frac{g}{c}$ holds. If $-\frac{e}{b}\neq -\frac{g}{c}$ then no solution exists. Otherwise, all possible solutions lie on a straight line within the $y=-\frac{e}{b}=-\frac{g}{c}$ plane that satisfies the equation

$$bx+cz+f=0 \tag{6.60}$$

which is derived from Equation 6.48. Analogously to Case 2.2, all locations along the straight line implied by Equation 6.60 are saddles of the bilinear interpolant on x- and z-axis perpendicular slices.

Case 2.5: $b=0; c=0; d\neq 0$
From Equation 6.48 it follows that $f=0$ must hold. If $f\neq 0$ then no solution exists. Otherwise, Equation 6.47 implies that $z=-\frac{e}{d}$ holds, and Equation 6.49 implies that $x=-\frac{g}{d}$ holds. All possible solutions lie on a straight line

$$x=-\frac{g}{d}\ ; \qquad y=t\in[0,1]\ \text{; and} \qquad z=-\frac{e}{d}\ . \tag{6.61}$$

Like in Case 1.5, this is an extended, degenerate criticality. If one considers y-axis perpendicular slices through the cell, the bilinear interpolant has a saddle at the intersection with the straight line implied by Equation 6.61, since $k_y = d \neq 0$. Since $a = 0$, $b = 0$, $c = 0$, and $f = 0$ it is possible to obtain values along the line implied by Equation 6.61 as

$$T\left(-\frac{g}{d}, t, -\frac{e}{d}\right) = \frac{deg}{d^2} - \frac{eg}{d} - \frac{eg}{d} + h = h - \frac{eg}{d} \; . \tag{6.62}$$

All bilinear saddles along the line have the same value.

Case 2.6: $b = 0; c \neq 0; d = 0$
From Equation 6.47 it follows that $e = 0$ must hold. If $e \neq 0$ then no solution exists. Otherwise, Equation 6.48 implies that $z = -\frac{f}{c}$ holds, and Equation 6.49 implies that $y = -\frac{g}{c}$ holds. All possible solutions lie on a straight line

$$x = t \in [0, 1] \; ; \qquad y = -\frac{g}{c} \; \text{; and} \qquad z = -\frac{f}{c} \; . \tag{6.63}$$

Like in Case 1.5, this is an extended, degenerate criticality. If one considers x-axis perpendicular slices through the cell, the bilinear interpolant has a saddle at the intersection with the straight line implied by Equation 6.63, since $k_x = c \neq 0$.

Case 2.7: $b \neq 0; c = 0; d = 0$
From Equation 6.49 it follows that $g = 0$ must hold. If $g \neq 0$ then no solution exists. Otherwise, Equation 6.47 implies that $y = -\frac{e}{b}$ holds, and Equation 6.48 implies that $x = -\frac{f}{b}$ holds. All possible solutions lie on a straight line

$$x = -\frac{f}{b} \; ; \qquad y = -\frac{e}{b} \; \text{; and} \qquad z = t \in [0, 1] \; . \tag{6.64}$$

Like in Case 1.5, this is an extended, degenerate criticality. If one considers z-axis perpendicular slices through the cell, the bilinear interpolant has a saddle at the intersection with the straight line implied by Equation 6.64, since $k_z = b \neq 0$.

Case 2.8: $b = 0; c = 0; d = 0$
From Equations (6.47, 6.48 and 6.49 it follows that $e = 0$, $f = 0$, and $g = 0$ holds. It follows that $T(x, y, z) = h$ holds. All positions within the cell have constant value, and no critical point exists within the cell.

Lemma 9 *Let C with vertex values A, B, C, D,A_1, B_1, C_1, and D_1, numbered as shown in 6.6. Let the face bounded by the vertices A, B, C, and D have a saddle with value zero, and let the cell be rotated in such a way that A and C are larger than zero. If the criterion from Lemma 5,*

i.e.,

$$connected_1 = \begin{cases} c_{1.1} & \text{if } c_{1.1} \neq 0 \\ c_{1.2} & \text{if } c_{1.1} = 0 \end{cases} \text{, with} \tag{6.65}$$
$$c_{1.1} = C(A_1 - A) + A(C_1 - C) - D(B_1 - B) - B(D_1 - D) \text{ , and}$$
$$c_{1.2} = (A_1 - A)(C_1 - C) - (B_1 - B)(D_1 - D) \text{ ,}$$

yields a value of zero indicating that neither positive or negative regions are connected in that cell, then the within the cell the gradient of the trilinear interpolant vanishes along a curve that contains this face saddle.

Proof:
Without loss of generality, assume that the cell is oriented in such a way that $A = v_0$, $B = v_3$, $C = v_7$, $D = v_4$, $A_1 = v_1$, $B_1 = v_2$, $C_1 = v_2$, and $D_1 = v_5$, *i.e.*, the "left" boundary face contains the saddle that is under consideration. The precondition states that the saddle value on the face $SV(0)$, see Equation 6.4, equals zero. Consequently, the numerator is zero, *i.e.*, $AC - BD = 0$. Using Equation 6.7 it is possible to express this requirement in terms of the coefficients of Equation 6.6 resulting in the equation

$$ch - fg = 0 \quad \Rightarrow \quad ch = fg \text{ .} \tag{6.66}$$

Since the saddle exists on the face, the denominator $A - B + C - D$ must differ from zero. From this requirement follows $c \neq 0$. If $connected_1$ is zero it follows that both $c_{1.1}$ and $c_{1.2}$ are zero. Expressing this in terms of the coefficients of Equation 6.4 yields

$$ah - bg - df + ce = 0 \text{ , and} \tag{6.67}$$
$$ae - bd = 0 \quad \Rightarrow \quad ae = bd \text{ ,} \tag{6.68}$$

respectively.

Case A: $a \neq 0$
From Equation 6.68 follows that $a_x = ae - bd$ equals zero. Considering the product $a_y a_z$ one obtains

$$\begin{aligned} a_y a_z =& (af - bc)(ag - cd) = a^2 \underbrace{fg}_{=ch \text{ (Equation 6.66)}} - acbg - acdf + c^2 \underbrace{bd}_{=ae \text{ (Equation 6.68)}} \\ =& ac \underbrace{(ah - bg - df + ce)}_{=0 \text{ (Equation 6.67)}} \text{ .} \end{aligned} \tag{6.69}$$

It follows that $a_y = 0$, or $a_z = 0$.

Case A.1: $a_y = 0$; $a_z \neq 0$
Since $a_x = 0$, the gradient behavior in the the cell's interior follows Case 1.5. The curve of bilinear saddles lies in the $z = -\frac{b}{a}$ plane that intersects the plane containing the saddle. The curve on that face is described by Equation 6.38. To show that the curve intersects the plane containing

the saddle it is necessary to show that it intersects the edge resulting from the intersection of the plane containing the bilinear saddles and the boundary curve. Within the $z = -\frac{b}{a}$ plane this is the edge connecting $\left(0, 1, -\frac{b}{a}\right)$ and $\left(0, 1, -\frac{b}{a}\right)$. The values at the locations are

$$\begin{aligned} C(0,0) =& g \text{ , and} \\ C(0,1) =& c + g \text{ .} \end{aligned} \tag{6.70}$$

From the precondition follows that $A = h > 0$, $B = f + h < 0$, $C = c + f + g + h > 0$, and $D = g + h < 0$. Thus, it follows that $g < -h < 0$ and $c + g > -(f + h) > 0$. It follows, that the curve containing all bilinear saddles indeed intersects the boundary face. Since a bilinear interpolant contains at most one saddle, this saddle coincides with the considered saddle and this saddle is connected to another face via an extended internal saddle.

Case A.2: $a_y \neq 0$; $a_z = 0$
Since $a_x = 0$, the gradient behavior in the cell's interior follows Case 1.6. The proof that the curve containing the bilinear saddles intersects the boundary face containing the considered saddle is analogous to Case A.1 and follows from symmetry.

Case A.3: $a_y = 0$; $a_z = 0$
Since $a_x = 0$, the gradient behavior in the cell's interior follows Case 1.8. All boundary faces are intersected by the three axis-aligned lines. Since the bilinear interpolant on the considered face has at most one saddle, this saddle is connected to the other faces via the three axis-aligned lines.

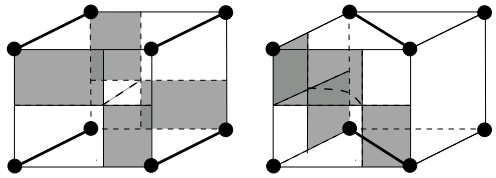
Case B: $a = 0$
Since a equals zero it follows from Equation 6.68 that bd equals zero. It follows that b, or d equals zero.

Case B.1: $b = 0$; $d \neq 0$
Since $c \neq 0$, the gradient behavior in the cell's interior follows Case 2.1. As a and b equal zero, it follows from Equation 6.67 that $ce - df$ equals zero. From this observation follows that $-\frac{e}{d} = -\frac{c}{f}$. All bilinear saddles lie on the line $L(x, y) = dx+cy+g = 0$ within the $z = -\frac{e}{d} = -\frac{f}{c}$ plane. To determine, whether it intersects the boundary face containing the saddle under consideration one can check the values at the ends of the line segment resulting from intersection the plane containing the line with the boundary face of the cell, *i.e.*, the edge connecting $\left(0, 1, -\frac{e}{d}\right)$ and $\left(0, 1, -\frac{e}{d}\right)$. It follows that $L(0, 0) = c + g > 0$ (see Case A.1) and $L(0, 1) = g < 0$ (see Case A.1). The line intersects the plane. Since the boundary face contains a unique saddle, it is connected to another face via an extended internal saddle.

Case B.2: $b \neq 0$; $d = 0$
Since $c \neq 0$, the gradient behavior in the cell's interior follows Case 2.4. The proof that the line of bilinear saddles exists an intersects the boundary face containing the considered saddle is analogous to Case B.1 and follows from symmetry.

Face number	Enclosing vertices
f_0	$v_0v_3v_2v_1$
f_1	$v_1v_2v_6v_5$
f_2	$v_4v_5v_6v_7$
f_3	$v_0v_4v_7v_3$
f_4	$v_0v_1v_5v_4$
f_5	$v_2v_3v_7v_6$

Table 6.1: Numbering scheme used for cell faces.



(a) (b)

Figure 6.29: Edges added between nodes representing vertices in connectivity graph. (a) The opposite face is saddle-connected. (b) A neighboring face is saddle-connected. A bold line indicates that an edge in the neighborhood graph is added between the nodes corresponding to the vertices.

Case B.3: $b = 0$; $d = 0$
Since $c \neq 0$, the gradient behavior in the cell's interior follows Case 2.6. As a, b, and d equal zero, it follows from Equation 6.67 that ce equals zero. Since c is not equal to zero, e must be zero and a solution for a line containing saddles of bilinear slices exist. This line connects the boundary face containing the considered saddle with the opposite cell face.

6.9.3 Classifying Extended Saddles

Extended interior saddles are classified by tracing them through cells and constructing a graph that represents connected positive and negative regions around them. Using an indexing scheme that effectively numbers all faces of a rectilinear grid a bit vector is maintained that contains a bit for each face of the grid. When set, this bit indicates that a potential saddle on the face has already been handled. The index for a face can be obtained by specifying a cell index and a face number within a cell, see Table 6.1. Faces that are shared between cells yield the same index in both cells. In addition, an indexing scheme that numbers all vertices of a rectilinear grid is employed. The vertices are used as nodes in the graphs corresponding positive and negative regions in the neighborhood around a saddle. Each extended interior saddle connects two or more faces of a cell. Each of these faces has a saddle, partitioning that face into four regions, each containing a face vertex. Consequently, those face regions can be represented by face vertices. In order to differentiate between connectivity graph nodes that correspond to vertices

and nodes that correspond to edges, positive integers are used for numbering edges, and negative integers are used for numbering vertices. If it is possible to classify a face saddle, locally, a flag is set, indicating that the corresponding face was handled. Otherwise, nodes corresponding to the positive and negative vertices of the face are added to the positive and negative neighborhood graph respectively, and the flag indicating that the face has been handled is set. Subsequently, the considered face is treated as two "half faces," *i.e.*, each side of it that lies in one of the two adjacent cells is considered independently, and the extended saddle is traced recursively in the corresponding direction.

First, it is checked if the face has already been handled during tracing an extended saddle by using the corresponding flag in the bit field. If the face has been handled, the recursion terminates. Otherwise, the corresponding flag is set. Using the criterion from Lemma 5 it is determined whether the positive or negative regions of the face saddle are connected in the considered cell. If positive or negative regions are connected, an edge is added between the graph nodes corresponding to the positive or negative vertices of the face containing the saddle, and the recursion terminates. Otherwise, according to Lemma 9, at least one saddle-connected face exists. This is either an unique cell face differing from the currently considered face (Cases 1.5– 1.7; Cases 2.2–2.7), or, all faces of the cell are saddle-connected (Case 1.8). If a unique saddle-connected face exists, its vertices are added to the neighborhood graph. Additionally, edges are added to both neighborhood graphs connecting the nodes corresponding to the positive and negative vertices of the face, reflecting the internal connections between the corresponding regions, see Figure 6.29. Two configurations are possible: A face can be connected to its opposite face, see Figure 6.29(a), or a neighboring face, see Figure 6.29(b). The corresponding face is handled recursively like the original face. If all faces are saddle-connected, the cell is partitioned into eight disjoint regions (four positive and four negative regions). Nodes for all vertices (but no connection between them) are added to the neighborhood graph and the saddle is recursively traced through all faces. When tracing the saddle terminates, the number of connected components in the resulting connectivity graphs are used, to determine whether the connected region is a saddle or a regular point.

6.9.4 Determining Saddle-connected Faces

The face containing the saddle is either connected to the opposite face, one neighboring face, or all other boundary faces (Case 1.8). Saddle connected faces are determined as follows: First, a distinction is made between case 1 and 2 by examining the value of a. If a differs from zero, a sub-case is determined based on the values of a_x, a_y, and a_z. By Lemma 9, one of the cases 1.5–1.8 arises.

For Case 1.5, the curve containing the saddles of bilinear slices lies in the $z = -\frac{b}{a}$ plane, see Figure 6.30. On this plane, the locations of saddles on bilinear slices through the cell are described by the bilinear contour given by Equation 6.38. Values at the vertices $v_0, \ldots, v_3$ of the rectangle, obtained by intersecting the $z = -\frac{b}{a}$ slice with the cell's boundary, are computed. These vertices have the positions $\left(0, 0, -\frac{b}{a}\right)$, $\left(1, 0, -\frac{b}{a}\right)$, $\left(1, 1, -\frac{b}{a}\right)$, $\left(0, 1, -\frac{b}{a}\right)$, respectively.

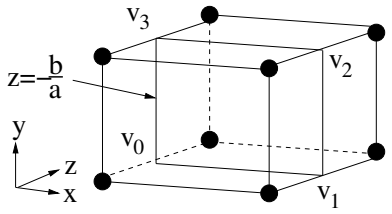


Figure 6.30: To compute interior connections for Case 1.5, a bilinear slice perpendicular to the z-axis is considered.

Values at these vertices are obtained as

$$\begin{aligned} v_0 &= C(0,0) = g \ , \\ v_1 &= C(1,0) = d + g \ , \\ v_2 &= C(1,1) = a + d + c + g \text{ , and} \\ v_3 &= C(0,1) = c + g \ . \end{aligned} \tag{6.71}$$

A case table approach is used to determine which edges of the bilinear slice are connected. A case number is computed as

$$\text{caseNo} = \sum_{i=0}^{3} \begin{cases} 0 & \text{if } v_i < 0 \\ 2^i & \text{if } v_i \geq 0 \end{cases} \ . \tag{6.72}$$

Using this case number, Table 6.2 is referenced to determine which edges of the bilinear slice are connected by the curve. For Cases 5 and 10 it is necessary to consider the value of the saddle of the bilinear interpolant, obtained by using Equation 2.40 to disambiguate between two possible connection configurations. The edges coincide with cell boundary faces, *i.e.*, edge $e_0, \ldots, e_3$, coincide with faces f_4, f_1, f_5, and f_3 respectively. Thus, it is possible to determine which faces are saddle-connected to the current face. Since neither positive nor negative regions are connected within the cell, a saddle-connected face must exit according to Lemma 9. As observed in Case 1.5, the degenerate case where the contour on the slices is a pair of axis aligned asymptotes does not occur. Consequently, the face containing the considered saddle has a unique saddle-connected face. Cases 1.6 and 1.7 are handled analogously to Case 1.5. For Case 1.8, all six faces are saddle-connected. This is the only case where more than two faces are saddle-connected.

When a equals zero, a sub-case is determined according to the values of b, c, and d. One of the Cases 2.2–2.7 must occur. For Cases 2.2–2.4, all saddle of bilinear slices lie on a line segment within a plane. As for Case 1.5, the values at the four vertices $v_0, \ldots, v_3$ where the bilinear slice intersects the cell boundaries are computed. As linear interpolation can be considered as a special case of bilinear interpolation, Table 6.2 can be used to determine which edges, and, in turn, which faces are saddle-connected. For Cases 2.5–2.7 a face is saddle connected to its opposite cell face.

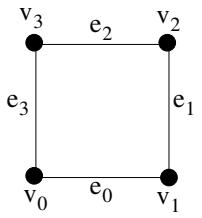


Figure 6.31: Vertex and edge numbering scheme employed in Table 6.2.

Case Number	Connected Edges
0	none
1	(e_0, e_3)
2	(e_0, e_1)
3	(e_1, e_3)
4	(e_1, e_2)
5	(e_0, e_3) and (e_1, e_2) if $sV < 0$; (e_0, e_1) and (e_2, e_3) is $sV > 0$
6	(e_0, e_2)
7	(e_2, e_3)
8	(e_2, e_3)
9	(e_0, e_2)
10	(e_0, e_3) and (e_2, e_3) if $sV < 0$; (e_0, e_1) and (e_1, e_2) is $sV > 0$
11	(e_1, e_2)
12	(e_1, e_3)
13	(e_0, e_1)
14	(e_0, e_3)
15	none

Table 6.2: Edge connection configurations for a bilinear slice. See Figure 6.31 for vertex and edge numbering convention.

6.10 Limitations

In addition to the problems encountered when classifying extended saddles, *i.e.*, classification of a larger region than the actual saddle as saddle region, merging of saddles and missing saddles whose positive *and* negative regions are connected by a regular region, see Section 6.7, the current implementation has a few limitations.

Most notably is the fact, that a constant classification region with non-trivial topology can imply a topology change, even though it is regular according to the Definition 10, *i.e.*, its neighborhood contains exactly one positive and one negative region. Cox *et al.* [15] referred to these constant valued regular regions with non-trivial topology as "critical-regular isosets." Figure 6.32 shows an example. If the torus corresponds to a "solid" region of constant value, it is a regular region according to Definition 10 as it is surrounded by one positive and one negative region.

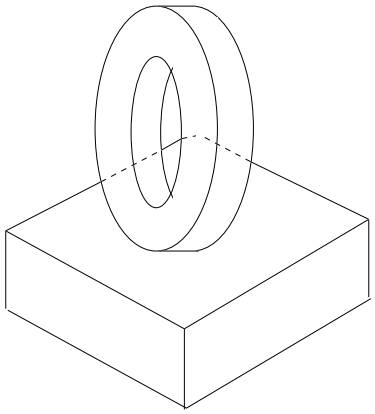


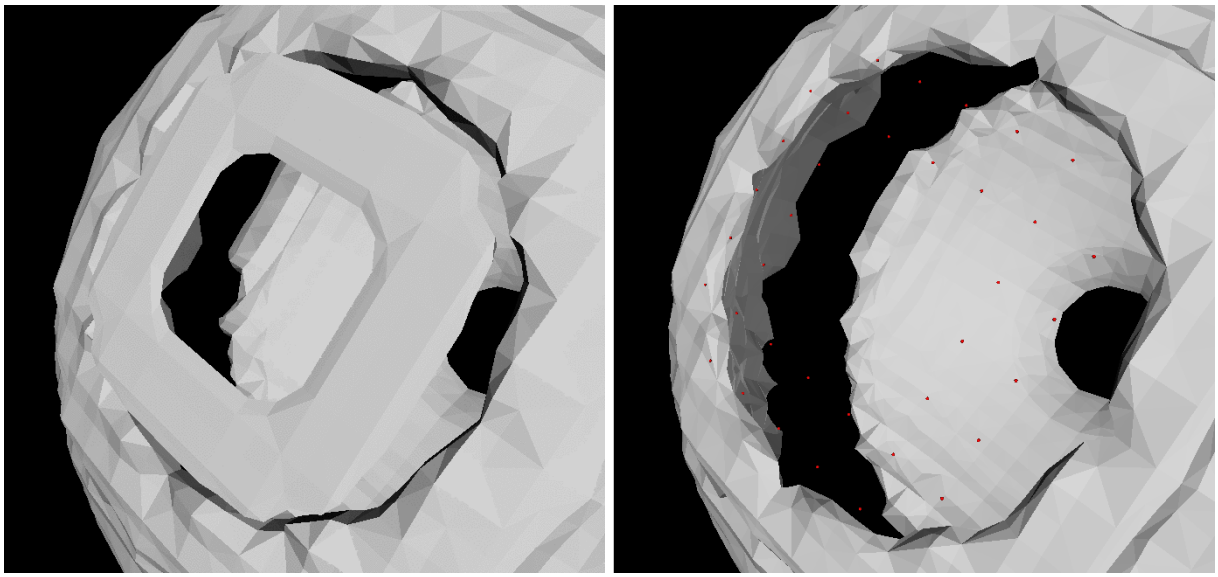
Figure 6.32: At a constant-valued regular region with non-trivial topolgy, topology of an isosurface can change.

Nonetheless, a genus-change occurs at the corresponding isovalue. My method does not detect regions with that property.

My method for tracing face saddles is susceptible to numerical problems due to floating-point inaccuracies. Furthermore, critical regions consisting of vertices and extended face saddles independently are classified. A special case arises, when Configuration $C8.3.2$ is encountered during the classification of a connected region of vertices. In this instance, a region of vertices is saddle connected to a boundary face that has a saddle with the same value as the classification region. This configuration can be handled by adding the vertices of the boundary face containing the saddle to the neighborhood graphs. A graph-edge must be added between each node corresponding to a vertex of the ambiguous face and the the node corresponding to the cell-edge connecting it to the classification region. Subsequently, the saddle can be traced as described above. While tracing this saddle, other cells that have a vertex configuration equivalent to Configuration $C8.3.2$ can be encountered. For each of these cells it would be necessary to identify the corresponding region of connected vertices and include it in the classification process. However, detecting Configuration $C8.3.2$ is extremely susceptible to numerical problems. I did not encounter a data set where it occurs, yet. Due to the lack of test data, I chose to ignore this configuration. For data sets where this configuration occurs, this limitation may lead to the incorrect classification of a regular region as saddle.

6.11 Data Exploration Using Critical Regions

Critical regions can be used to explore volumetric data in the same way as critical point. The "isovalue navigator" window is extended to list region criticalities (region minimum, region maximum, region saddle, extended face saddle). When a user selects a critical point, its corresponding position in space is marked by a sphere whose color depends on the type (blue, red, and green representing minimum, maximum, or saddle, respectively). For a region, its position is marked



(a) (b)

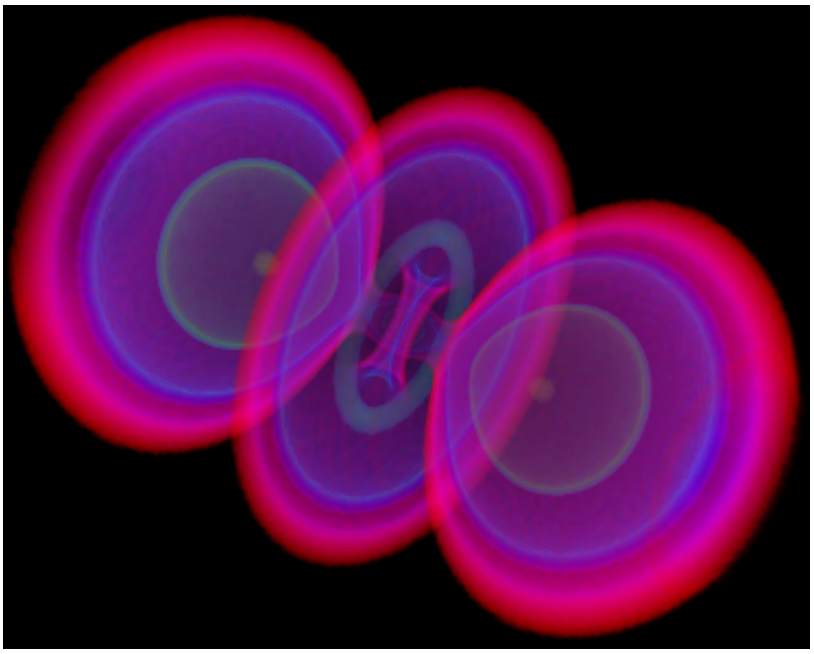
Figure 6.33: Region maximum in “Nucleon” data set. (Data set courtesy of SFB 382 of the German Research Council (DFG)). (a) Isosurface for an isovalue slightly below the maximum. (b) Isosurface for an isovalue slightly above the maximum.

by a point set of all vertices belonging to the critical region with a color corresponding to type. When a user wants to center the view on a critical region, its centroid is used as camera focus. Transfer functions can be constructed automatically using the unmodified scheme that is used for critical points, since the transfer function design scheme only uses values of critical points.

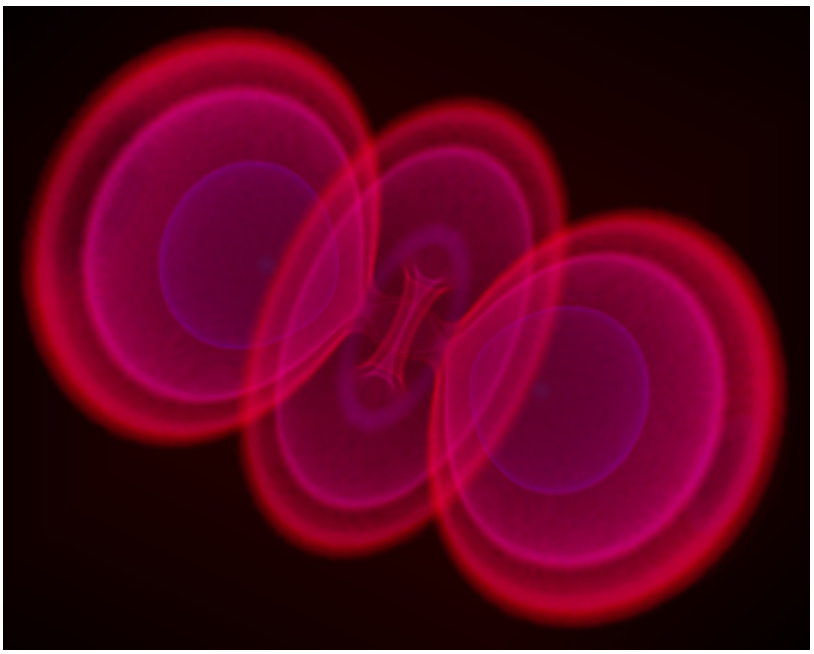
6.12 Results of Critical Region Analysis

Figure 6.33 shows a region maximum of value 190 in the “Nucleon” data set (Data set courtesy of SFB 382 of the German Research Council (DFG), available at `http://www.volvis.org`) which was missed by the previous approach [85]. In the neighborhood of this maximum a torus-shaped isosurface component disappears.

Figures 6.34 and 6.35 shows results obtained by applying topological analysis to the “Hydrogen” data set (Data set courtesy of SFB 382 of the German Research Council (DFG), available at `http://www.volvis.org`). Figure 6.34 shows volume-rendered images of the data set using transfer functions that were automatically generated from a list of critical isovalues. Figures 6.35 (a) – (j) show a topological analysis of the same data set using isosurfaces.



(a)



(b)

Figure 6.34: Volume rendered images of the “Hydrogen” data set with automatically generated transfer functions emphasizing topological changes (a) and zones of similar topological behavior (b).

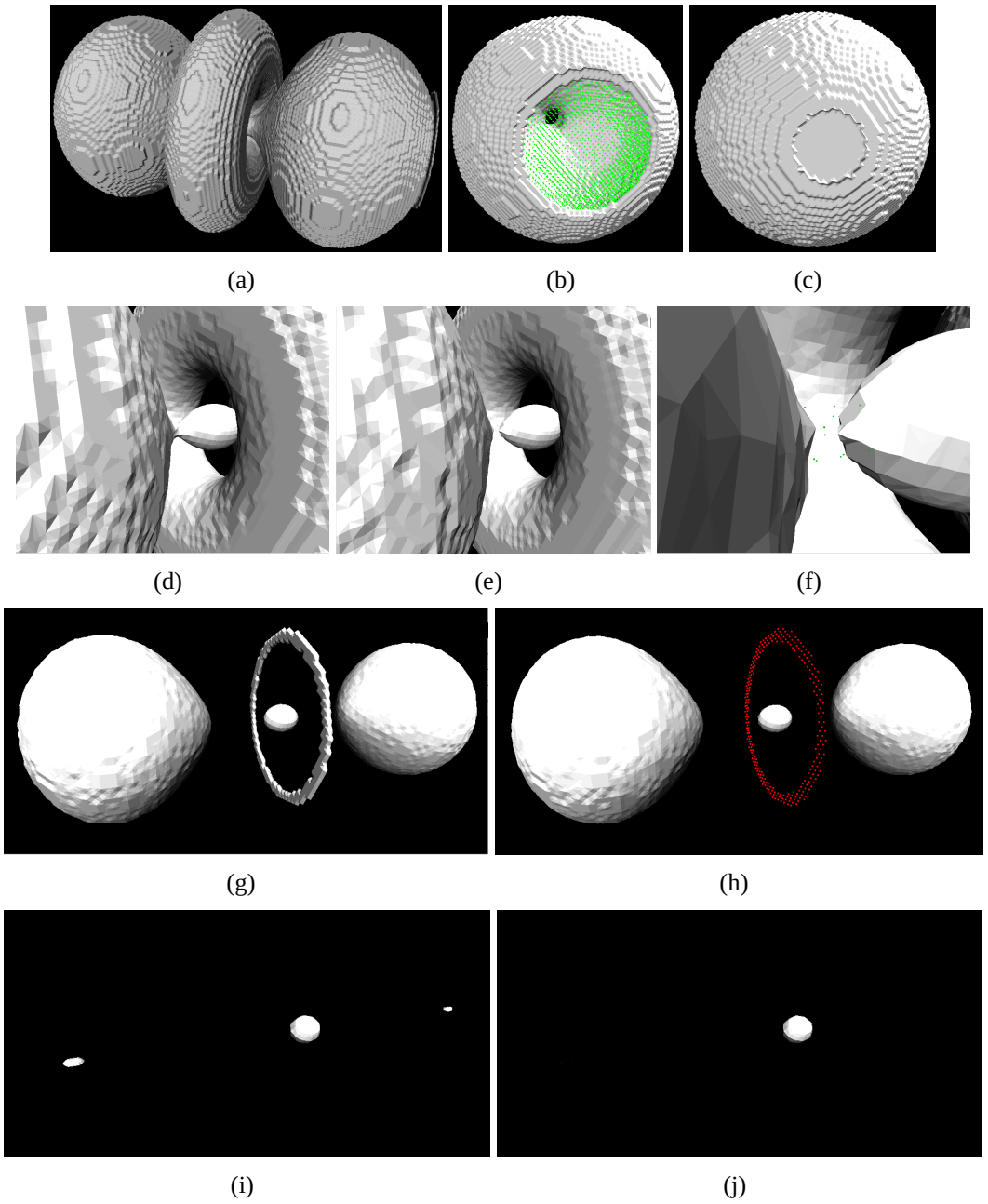


Figure 6.35: Topological analysis of the "Hydrogen" data set: (a) – (j) topological structure; (a) around a region minim having a value of zero, two components appear simultaneously; (b), (c) at a saddle region having a value of 5, a hole in one surface component closes; (d), (e) the "inner" surface is separated into three disjoint components along two saddle regions having a value of 12; (f) a close-up of one of the two saddle regions having a value of 12; (g), (h) The "ring" component disappears around a region maximum having a value of 36; (i),(j) Two components disappear at two region maxima having a value of 80; The remaining component disappears around a localized maximum having a value of 250. (Data set courtesy of SFB 382 of the German Research Council (DFG))

Chapter 7

Conclusions and Future Work

7.1 Visualization of AMR Data

7.1.1 Crack-free Isosurfaces for AMR Data

I have presented a method for the extraction of crack-free isosurfaces from AMR data. By using a dual-grid approach and filling gaps with stitch cells I avoid re-sampling of data and dangling nodes. By extending the standard MC method to the cell types resulting from grid stitching, I have developed an isosurfacing scheme that produces consistent and seamless isosurfaces. Theoretically, any continuous (scattered data) interpolation scheme could also lead to a crack free isosurface. However, the use of dual grids and stitch cells ensures that the resolution of extracted isosurfaces automatically varies with the resolution of the data given in a region.

Several extensions to the index-based stitching scheme are possible. The original AMR scheme by Berger and Colella [6] requires a layer of width of at least one grid cell between a refining grid and the boundary of a refined level. Even though Bryan [8] eliminates this requirement in his simulation, the stitching approach depends on the existence of a boundary layer of unrefined cells. This layer is necessary to ensure that only transitions between a coarse level and the next finer level occur in an AMR hierarchy. Allowing transitions between arbitrary levels would require the consideration of too many cases during the stitching process. (The number of cases would be limited only by the number of levels in an AMR hierarchy, since within this hierarchy transitions between arbitrary levels are possible.) Unfortunately, this requirement does not allow handling the full range of AMR data sets in use today, *e.g.*, those produced by the methods of Bryan [8]. It may be possible to modify the case table approach to detect arbitrary transitions and use a more generic stitch cell generation scheme for these transitions. It should also be possible to use this approach for AMR hierarchies containing rotated grids, *e.g.*, data sets produced by the AMR method of Berger and Oliger [7]. When grids in a child level are rotated only up to a certain maximum rotation, stitch cells are deformed but remain a valid tessellation of the domain. By modifying the connection scheme to locate appropriate coarse-grid cells during stitching, it may be possible to extend the stitching scheme to Berger-Oliger AMR data. It would also be interesting to use a generic triangulation scheme, like Delaunay-tetrahedrization to fill gaps between grids and compare results to my procedural stitching scheme. However, this would

require the replacement of adjacent grid cells in the regular, rectilinear grids with subdivisions into tetrahedra and pyramids to ensure that no quadrilateral boundary face of a cell is subdivided by tetrahedra of the stitch-mesh. Alternatively, it may be possible to define a consistent interpolation scheme based on the work of Forsey and Bartels [23] and derive an isosurface extraction method based on that scheme.

For transforming of level coordinates to grid coordinates, currently each grid in a level is examined whether it contains a given grid point. This is sufficiently efficient for moderately sized data sets; but for larger data sets, a space subdivision-based search scheme should be used. It should be possible to use a modification of the generalized k-d trees from my homogenization scheme for hardware-accelerate rendering for this purpose.

7.1.2 Volume Rendering of AMR Data

Hardware-accelerated Volume Rendering

By utilizing the inherent hierarchical structure of AMR data, I can improve efficiency of AMR data visualization substantially. By homogenizing an AMR data set, it becomes possible to use standard techniques for hardware-accelerated volume rendering. Most work that followed the publication of my method [80] uses some kind of homogenization for volume rendering of AMR data sets. I have also presented a hardware-accelerated approach that simulates the cell projection method. Given the recent rapid developments in hardware-accelerated volume rendering, it would be interesting to replace it with a more current hardware-accelerated scheme. With modern graphics hardware it may even be possible to develop a new approach that simulates cell-projection at higher quality avoiding visible artifacts. Furthermore, the AMR hierarchy can be used to guide a viewer for navigation through a data set and/or render only regions of interest. My approach of adapting the traversal depth tends to oscillate. More work based, for example, on Meyer [63] is necessary to reduce this effect. Combining viewpoint-dependent and framerate-dependent criteria could assign priorities to different nodes and determine regions where quality is to be sacrificed for rendering speed, yielding a more flexible, adaptive traversal strategy for AMR partition trees.

7.1.3 Progressive Cell-projection Method for AMR Data

My high-quality cell-projection rendering method allows me to progressively render the refining grids, producing images of increasing quality. By using cell-projection, my approach automatically adapts the sampling rate to the local resolution. Improvements are possible concerning the quality of produced images and efficiency of the algorithm. The generation of ray segments during cell projection can be improved. It may be possible to describe the contribution of a ray-segment within a cell analytically, or develop an analytical approximation that produces better quality than numerical integration. On the other hand, more advanced lighting models and numerical integration schemes can be used. Furthermore, alternative interpolation functions for pyramid cells should be explored.

I have reimplemented the cell-projection scheme more efficiently for my work on parallel volume rendering of AMR data. Even though this reimplementation currently only supports constant interpolation it leads to several improvements that can be used for a high-quality rendering scheme. First, the ability to progressively refine images imposes a considerable overhead in image generation. Given the existence of powerful hardware-accelerated volume rendering schemes for preview rendering, a future implementation should discard that ability. The main performance impact results from sorting ray-segments instead of cells and checking whether ray-segments are from different levels. An efficient cell-projection scheme for AMR data should pre-sort cells and composite ray-segment contributions immediately.

7.1.4 Parallel Rendering of AMR Data

I have implemented and compared several distribution strategies for direct volume rendering of AMR hierarchies. Homogeneous subdivision supports efficient rendering of AMR data for different classes of machines. It allows me to avoid data duplication and employ a wide variety of rendering schemes. Homogenizing an AMR hierarchy has also been used for a variety of hardware-accelerated methods for volume-rendering AMR data. While weighted homogeneous subdivision of the domain results in near-uniform processor utilization, I plan to improve the approximation of relative cell weights. In particular it may be beneficial to use view-dependent weights. I also intend to consider inhomogeneous PC clusters consisting of machines with varying processor speed and develop subdivision/distribution methods that take differences in machine performance into account. Furthermore, I plan to develop a communication-less subdivision strategy that avoids the need for each processor to compute assignments for all other processors, resulting in less overhead and better scalability. During compositing, a major portion of time is spent on sending and receiving partial images. I plan to reduce this overhead by encoding partial images more efficiently using, for example, run-length encoding. Furthermore, I intend to implement binary-swap compositing [59] and compare its results to mine.

7.2 Topology-based Scalar Data Analysis and Visualization

I have presented a method for the detection and utilization of critical isovalues for the exploration of trivariate scalar fields defined by piecewise trilinear functions. My isosurface navigator allows a user to examine the topological structure of a data set and is valuable tool in volume data exploration. I have also applied my critical point analysis toward the automatic generation of transfer functions. Resulting volume rendered images reveal the underlying topological structure of a volume data set.

Some data sets contain a large number of critical points. Some of these critical points correspond to locations/regions of actual interest, but some are the result of noise or improper sampling. I need to develop methods to eliminate such “false” critical points. On the other hand it could be useful to consider more noisy data sets and generate a histogram with the number of topology changes for a lot of small isovalues ranges. It should be possible to automatically detect interesting isovalues by looking for values where there are many topological changes. This

could be used to detect turbulence in data sets resulting from unsteady flow simulations in which turbulence is usually associated to "topological noise." Histograms could also be used to generate meaningful transfer functions for data sets with a large number of closely spaced critical isovalues. Developing additional automatic transfer function schemes could also help in gaining insight in data sets containing a large number of critical regions/points and isovalues.

I have also extended my algorithm to detect critical regions in volume data sets. My approach detects exact regions for maxima and minima. Region saddles are detected in most cases. However, my approach can mark a large region as a saddle, a region larger than the actual saddle region. It may also "merge" several isolated saddles into a saddle compound, when saddles are connected via a set of vertices of the same value as the saddle. I plan to extend my approach to detect exact regions of saddles. This approach would also eliminate the problem that saddles can be missed when a regular region connected both its positive and negative regions. To achieve this goal, it may be best to merge the flood-fill pass with the graph-construction pass, constructing the graph while vertices of constant value are traced, which would simplify merging the handling of extended face saddles with classifying regions consisting of grid vertices of constant value. It should also be possible to adapt my approach to detect critical regions for piecewise linear simplicial meshes. Instead of treating critical points individually it may prove beneficial to merge my approach with the work of Pascucci and Cole-McLaughlin [71]. Contour trees establish a relationship between critical points which would be helpful in simplifying the topology of a volume data set and removing topological noise. If a classification region has non-trivial topology, surface components with a genus larger than zero can appear. A possible extension to my method could determine the topology of a classification region and determine the genus of the newly created component. This extension may also allow for the detection of "regular regions" where isosurface topology changes, see Section 6.10.

Bibliography

[1] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. The contour spectrum. In: Roni Yagel and Hans Hagen, editors, *IEEE Visualization '97*, pages 167–173, IEEE, ACM Press, New York, New York, October 19–24 1997.

[2] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. Visualizing scalar topology for structural enhancement. In: David S. Ebert, Holly Rushmeier, and Hans Hagen, editors, *IEEE Visualization '98*, pages 51–58, IEEE, ACM Press, New York, New York, October 18–23 1998.

[3] Chandrajit L. Bajaj and Daniel R. Schikore. Topology preserving data simplification with error bounds. *Computers & Graphics*, 22(1):3–12, 1998.

[4] Thomas F. Banchoff. Critical points and curvature for embedded polyhedral surfaces. *American Mathematical Monthly*, 77(5):475–485, May 1970.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[6] Marsha Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, May 1989. Lawrence Livermore National Laboratory, Technical Report No. UCRL-97196.

[7] Marsha Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, March 1984.

[8] Greg L. Bryan. Fluids in the universe: Adaptive mesh refinement in cosmology. *Computing in Science and Engineering*, 1(2):46–53, March/April 1999.

[9] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: *Proceedings of the 1994 Symposium on Volume Visualization*, pages 91–98, ACM Press, New York, New York, 1994.

[10] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Computational Geometry – Theory and Applications*, 24(2):75–94, February 2003.

[11] Evgeni V. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Technical Report CN/95-17, CERN, Geneva, Switzerland, 1995. Available as `http://wwwinfo.cern.ch/asdoc/psdir/mc.ps.gz`.

[12] L. Paul Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.

[13] H. N. Christiansen and T.W. Sederberg. Conversoin of complex contour lines definitions into polygonal element mosaics. *Computer Graphics (Proceedings of ACM SIGGRAPH 78)*, 14(3):187–192, 1978.

[14] Paulo Cignoni, Fabio Ganovelli, Claudio Montani, and Roberto Scopigno. Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computers & Graphics*, 24(3):399–418, June 2000.

[15] Jim Cox, Daniel B. Karron, and Nazma Ferdous. Digital morse theory for scalaer volume data. Technical report, Computer Aided Surgery, Inc., 2002. Available online at `http://www.casi.net/DMT.pdf`.

[16] Thomas W. Crockett. An introduction to parallel rendering. *Parallel Computing*, 23(7):819–843, July 1997.

[17] Martin J. Dürst. Additional reference to "marching cubes" (letters to the editor). *Computer Graphics*, 22(2):72–73, April 1988.

[18] Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valeria Pascucci. Morse-smale complexes for piecewise linear 3-manifolds. In: *Proceedings of the 19th Annual ACM Symposum on Computational Geometry*, 2003. To appear.

[19] Herbert Edelsbrunner, John Harer, and Afra Zomorodian. Hierarchical morse complexes for piecewise linear 2-manifolds. In: *Proceedings of the 17th Annual ACM Symposium on Computational Geometry*, pages 70–79, 2001.

[20] Thomas Ertl, Rüdiger Westerman, and Roberto Grosso. Efficiently using graphics hardware in volume rendering applications. *Proceedings of ACM SIGGRAPH 98*, pages 169–177, July 1998.

[21] James D. Foley, Andries Van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principle and Practice*, second edition in c edition. Addison-Wesley, July 1995.

[22] Thomas A. Foley and Hans Hagen. Advances in scattered data interpolation. *Surveys on Mathematics for Industry*, 4(2):71–84, 1994.

[23] David R. Forsey and Richard H. Bartels. Hierarchical b-spline refinement. *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, 22(4):205–212, August 1988.

[24] Issei Fujishiro, Taeko Azuma, and Yuriko Takeshima. Automating transfer function design for comprehensible volume rendering based on 3D field topology analysis. In: David S. Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 467–470, IEEE, IEEE Computer Society Press, Los Alamitos, California, October 25–29, 1999.

[25] Issei Fujishiro and Yuriko Takeshima. Solid fitting: Field interval analysis for effective volume exploration. In: Hans Hagen, Gregory M. Nielson, and Frits Post, editors, *Scientific Visualization Dagstuhl '97*, pages 65–78, IEEE, IEEE Computer Society Press, Los Alamitos, California, June 1997.

[26] Issei Fujishiro, Yuriko Takeshima, Taeko Azuma, and Shigeo Takahashi. Volume data mining using 3D field topology analysis. *IEEE Computer Graphics and Applications*, 20(5):46–51, September/October 2000.

[27] Allen Van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, October 1994.

[28] Thomas Gerstner and Renato Pajarola. Topology preserving and controlled topology simplifying multiresolution isosurface extraction. In: Thomas Ertl, Bernd Hamann, and Amitabh Varshney, editors, *IEEE Visualization 2000*, pages 259–266, 565, IEEE, IEEE Computer Society Press, Los Alamitos, California, 2000.

[29] Dan Gordon, Michael A Peterson, and R. Anthony Reynolds. Fast polygon scan conversion with medical applications. *IEEE Computer Graphics and Applications*, 14(6):20–27, November 1994.

[30] Markus H. Gross, Oliver G. Staadt, and Roger Gatti. Efficient triangular surface approximations using wavelets and quadtree data structures. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):130–143, June 1996.

[31] Bernd Hamann. *Visualization and Modeling Contours of Trivariate Functions*. Ph.D. dissertation, Department of Computer Science, Arizona State University, Tempe, Arizona, USA, May 1991. Available at `http://graphics.cs.ucdavis.edu/~hamann/hamann.html`.

[32] Bernd Hamann. Modeling contours of trivariate data. *Mathematical Modeling and Numerical Analysis*, 26(1):51–75, 1992.

[33] Bernd Hamann, Issac J. Trotts, and Gerald E. Farin. On approximating contours of the picewise trilinear interpolant using triangular rational-quadratic Bézier patches. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):215–227, July-September 1997.

[34] John C. Hart. Morse theory for computer graphics. Technical Report EECS97-002, School of Electrical Engineering and Computer Science, Washington State University, May 1997. Appears in SIGGRAPH '97 Course Notes #14 "New Frontiers in Modeling and Texturing.".

[35] Klaus Jänich. *Topologie*, 4th edition. Springer Verlag, Berlin/Heidelberg, Germany, 1994.

[36] Ralf Kähler, Donna Cox, Robert Patterson, Stuart Levy, Hans-Christian Hege, and Tom Abel. Rendering the first star in the universe – a case study. In: Robert J. Moorhead, Markus Gross, and Kenneth I. Joy, editors, *IEEE Visualization 2002*, pages 537–540, IEEE, IEEE Computer Society Press, Los Alamitos, California, 2002.

[37] Ralf Kähler and Hans-Christian Hege. Interactive volume rendering of adaptive mesh refinement data. Technical Report ZR-01-30, Zuse Institur Berlin (ZIB), Berlin, Germany, 2001. Appeared in *The Visual Computer* [38]. Available as `ftp://ftp.zib.de/pub/zib-publications/reports/ZR-01-30.pdf`.

[38] Ralf Kähler and Hans-Christian Hege. Texture-based volume rendering of adaptive mesh refinement data. *The Visual Computer*, 18(8):481–492, 2002.

[39] Ralf Kähler, Mark Simon, and Hans-Christian Hege. Fast volume rendering of sparse high-resolution datasets using adaptive mesh refinement hierarchies. Technical Report ZR-01-25, Zuse Institur Berlin (ZIB), Berlin, Germany, 2001. Appeared in IEEE Transactions on Visualization and Computer Graphics [40]. Available as `ftp://ftp.zib.de/pub/zib-publications/reports/ZR-01-25.pdf`.

[40] Ralf Kähler, Mark Simon, and Hans-Christian Hege. Interactive volume rendering of large sparse data sets using adaptive mesh refinement hierarchies. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):341–351, July–September 2003.

[41] Daniel Karron. The "spiderweb" algorithm for surface construction in noisy volume data. In: Richard A. Robb, editor, *Proceedings of the SPIE (Visualization in Biomedical Computing 1992)*, volume 1808, pages 462–476, SPIE, SPIE – The International Society for Optical Engineering, Bellingham, WA, September 1992.

[42] Daniel B. Karron. New findings from the spiderweb algorithm: Toward a digital morse theory. In: Richard A. Robb, editor, *Proceedings of the SPIE (Visualization in Biomedical Computing 1994)*, volume 2359, pages 643–657, SPIE, SPIE – The International Society for Optical Engineering, Bellingham, WA, September 1994.

[43] Arie Kaufman. Efficient algorithms for scan-converting 3d polygons. *Computers & Graphics*, 12(2):213–219, 1988.

[44] Donald E. Knuth. *The Art of Computer Programming*, volume 1/Fundamental Algorithms, 3rd edition. Addison Wesley, 1997.

[45] Shirley Ellen Konkle, Patrick Moran, Bernd Hamann, and Kenneth I. Joy. Fast methods for computing isosurface topology with betti numbers. In: Frits Post, Gregory M. Nielson, and Georges-Pierre Bonneau, editors, *Data Visualization: The State of the Art-Proceedings Dagstuhl Seminar on Scientific Visualization*, pages 363–375. Kluwer Academic Publishers, Norwell, Massachusetts, 2003.

[46] Martin Kraus and Thomas Ertl. Topology-guided downsampling. In: Klaus Müller and Arie E. Kaufman, editors, *Proceedings of International Workshop on Volume Graphics '01*, pages 129–147, IEEE, IEEE Computer Society Press, Los Alamitos, California, 2000.

[47] Oliver Kreylos, Gunther H. Weber, E. Wes Bethel, John M. Shalf, Bernd Hamann, and Kenneth I. Joy. Remote interactive direct volume rendering of amr data. Technical Report LBNL 49954, Lawrence Berkeley National Laboratory, 2002.

[48] Philippe Lacroute. Fast volume rendering using a shear-warp factorization of the viewing transformation. Ph.D. dissertation CSL-TR-95-678, Stanford University, Computer Science Department, Computer Systems Laboratory, September 1995.

[49] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In: *Proceedings of ACM SIGGRAPH 94*, pages 451–458, ACM, New York, New York, July 1994.

[50] David Laur and Pat Hanrahan. Hierachical splatting: A progressive refinement algorithm for volume rendering. *Computer Grahpics (Proceedings of ACM SIGGRAPH 91)*, 25(4):285–288, July 1991.

[51] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988. (See also corrigendum [52, 91]).

[52] Marc Levoy. Letter to the editor: Error in volume rendering paper was in exposition only. *IEEE Computer Graphics and Applications*, 20(4):6–6, July/August 2000.

[53] Terry J. Ligocki, Brian Van Straalen, John M. Shalf, Gunther H. Weber, and Bernd Hamann. A framework for visualizing hierarchical computations. In: Gerald Farin, Bernd Hamann, and Hans Hagen, editors, *Hierarchical and Geometrical Methods in Scientific Visualization*, pages 197–204. Springer Verlag, Heidelberg, Germany, January 2003.

[54] Adriano Lopes and Ken Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):16–29, January–March 2003.

[55] Adriano M. Lopes. *Accuracy in Scientific Visualization*. Ph.D. dissertation, University of Leeds, United Kingdom, March 1999. Available at `http://www.mat.uc.pt/~adriano/Publications/thesis.ps.gz`.

[56] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 21(4):163–169, July 1987.

[57] Kwan-Liu Ma. Parallel rendering of 3D AMR data on the SGI/Cray T3E. In: *Proceedings of Frontiers '99 the Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 138–145, IEEE, IEEE Computer Society Press, Los Alamitos, California, February 1999.

[58] Kwan-Liu Ma and Thomas W. Crockett. A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data. In: James Painter, Gordon Stoll, and Kwan-Liu Ma, editors, *IEEE Parallel Rendering Symposium*, pages 95–104, IEEE, IEEE Computer Society Press, Los Alamitos, California, November 1997.

[59] Kwan-Liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. Parallel volume rendering using binary-swap composition. *IEEE Computer Graphics and Applications*, 14(2):59–67, July 1994.

[60] Peter MacNeice, Kevin M. Olson, Clark Mobarry, Rosalinda de Fainchtein, and Charles Packer. Paramesh: A parallel adaptive mesh refinement community toolkit. *Computer Physics Communications*, 126(3):330–354, April 2000.

[61] Nelson L. Max. Sorting for polyhedron compositing. In: Hans Hagen, Heinrich Müller, and Gregory M. Nielson, editors, *Focus on Scientific Visualization*, pages 259–268. Springer-Verlag, New York, New York, 1993.

[62] Nelson L. Max. Optical models for volume rendering. *IEEE Transactions on Computer Graphics*, 1(2):99–108, 1995.

[63] Jörg Meyer. *Interactive Visualization of Medical and Biological Data Sets*. Shaker Verlag, P.O. Box 1290, D-52013 Aachen, Germany, 2000. (Ph.D. disseration, AG Graphische Datenverarbeitung und Computergeometrie, Department of Computer Science, University of Kaiserlautern, Germany, 1999).

[64] John W. Milnor. *Morse Theory*. Princeton University Press, Princeton, New Jersey, May 1963.

[65] Claudio Montani, Riccardo Scateni, and Roberto Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer*, 10(6):353–355, 1994.

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

[67] Gregory M. Nielson. On marching cubes. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):341–351, July–September 2003.

[68] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. In: Gregory M. Nielson and Larry J. Rosenblum, editors, *IEEE Visualization '91*, pages 83–91, IEEE, IEEE Computer Society Press, Los Alamitos, California, 1991.

[69] Michael L. Norman, John M. Shalf, Stuart Levy, and Greg Daues. Diving deep: Data management and visualization strategies for adaptive mesh refinement simulations. *Computing in Science and Engineering*, 1(4):36–47, July/August 1999.

[70] Sanghun Park, Chandrajit Bajaj, and Vinay Siddavanahalli. Case study: Interactive rendering of adaptive mesh refinement data. In: Robert J. Moorhead, Markus Gross, and Kenneth I. Joy, editors, *IEEE Visualization 2002*, pages 521–524, IEEE, IEEE Computer Society Press, Los Alamitos, California, 2002.

[71] Valerio Pascucci and Kree Cole-McLaughlin. Efficient computation of the topology of level sets. In: Robert J. Moorhead, Markus Gross, and Kenneth I. Joy, editors, *IEEE Visualization 2002*, pages 187–194, IEEE, IEEE Computer Society Press, Los Alamitos, California, 2002.

[72] Thomas Porter and Tom Duff. Compositing digital images. *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, 18(3):253–259, July 1984.

[73] Paolo Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, 22(4):51–58, 1988.

[74] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. *The Visualization Toolkit*, second edition. Prentice-Hall, Upper Saddle River, New Jersey, 1998. With special contributors Lisa Sobierajski Avila, Rick Avila, and C. Charles Law (Includes CD-ROM with vtk-2.0. The most recent release is available on the World-Wide Web at `http://www.kitware.com/vtk.html`).

[75] Robert Sedgewick. *Algorithms in C++*, 2nd edition. Addison Wesley, April 1992.

[76] Raj Shekhar, Elias Fayyad, Roni Yagel, and J. Fredrick Cornhill. Octree-based decimation of marching cubes surface. In: Roni Yagel and Gregory M. Nielson, editors, *IEEE Visualization '96*, pages 335–342, 499, IEEE, IEEE Computer Society Press, Los Alamitos, California, October 1996.

[77] Allen Van Gelder and Kwansik Kim. Direct volume rendering with shading via three-dimensional textures. In: Roger Crawfis and Charles Hansen, editors, *1996 Volume Visualization Symposium*, pages 23–30, ACM Press, New York, New York, October 1996.

[78] Marc van Kreveld, René van Oostrum, Chandrajit Bajaj, Valerio Pascucci, and Daniel Osvaldo. Contour trees and small seed sets for isosurface traversal. In: Jean-Daniel Boissonnat, editor, *Proceedings of the Thirteenth ACM Symposium on Computational Geometry*, pages 212–219, ACM Press, New York, New York, June 4–6 1997.

[79] Bram van Leer. Towards the ultimate conservative difference scheme. IV. A new approach to numerical convection. *Journal of Computational Physics*, 23:276–299, March 1977.

[80] Gunther H. Weber, Hans Hagen, Bernd Hamann, Kenneth I. Joy, Terry J. Ligocki, Kwan-Liu Ma, and John M. Shalf. Visualization of adaptive mesh refinement data. In: Robert F. Erbacher, Philip C. Chen, Jonathan C. Roberts, Craig M. Wittenbrink, and Matti Groehn, editors, *Proceedings of the SPIE (Visual Data Exploration and Analysis VIII, San Jose, CA, USA, Jan 22–23)*, volume 4302, pages 121–132, SPIE, SPIE – The International Society for Optical Engineering, Bellingham, WA, January 2001.

[81] Gunther H. Weber, Oliver Kreylos, Terry J. Ligocki, John M. Shalf, Hans Hagen, Bernd Hamann, and Kenneth I. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In: David S. Ebert, Jean M. Favre, and Ronny Peikert, editors, *Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization, Ascona, Switzerland, May 28–31, 2001*, pages 25–34, 335, Springer Verlag, Wien, Austria, May 2001.

[82] Gunther H. Weber, Oliver Kreylos, Terry J. Ligocki, John M. Shalf, Hans Hagen, Bernd Hamann, and Kenneth I. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In: Gerald Farin, Bernd Hamann, and Hans Hagen, editors, *Hierarchical and Geometrical Methods in Scientific Visualization*, pages 19–40. Springer Verlag, Heidelberg, Germany, January 2003.

[83] Gunther H. Weber, Oliver Kreylos, Terry J. Ligocki, John M. Shalf, Hans Hagen, Bernd Hamann, Kenneth I. Joy, and Kwan-Liu Ma. High-quality volume rendering of adaptive mesh refinement data. In: Thomas Ertl, Bernd Girod, Günther Greiner, Heinrich Niemann, and Hans-Peter Seidel, editors, *Vision, Modeling, and Visualization 2001*, pages 121–128, 522. Akademische Verlagsgesellschaft Aka GmbH, Berlin, Germany and IOS Press BV, Amsterdam, Netherlands, November 2001.

[84] Gunther H. Weber, Martin Öhler, Oliver Kreylos, John M. Shalf, E. Wes Bethel, Bernd Hamann, and Gerik Scheuermann. Parallel cell projection rendering of adaptive mesh refinement data. In: Anton Koning, Raghu Machiraju, and Claudio T. Silva, editors, *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003*, pages 51–60, IEEE, IEEE Computer Society Press, Los Alamitos, California, October 2003.

[85] Gunther H. Weber, Gerik Scheuermann, Hans Hagen, and Bernd Hamann. Exploring scalar fields using critical isovalues. In: Robert J. Moorhead, Markus Gross, and Kenneth I. Joy, editors, *IEEE Visualization 2002*, pages 171–178, IEEE, IEEE Computer Society Press, Los Alamitos, California, 2002.

[86] Gunther H. Weber, Gerik Scheuermann, and Bernd Hamann. Detecting critical regions in scalar fields. In: Georges-Pierre Bonneau, Stefanie Hahmann, and Charles D. Hansen, editors, *Data Visualization 2003 (Proceedings of VisSym 2003)*. EUROGRAPHICS and IEEE TCVG, 2003. Accepted for publication.

[87] Eric W. Weisstein. Eric weisstein's world of mathematics. Available online at http://mathworld.wolfram.com/.

[88] Rüdiger Westermann, Leif Kobbelt, and Thomas Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.

[89] Lee Westover. Footprint evaluation for volume rendering. *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, 24(4):367–376, August 1990.

[90] Peter L. Williams, Nelson L. Max, and Clifford M. Stein. A high accuracy volume renderer for unstructured data. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):37–54, January-March 1998.

[91] Craig Wittenbrink, Tom Malzbender, and Mike Goss. Letter to the editor: Interpolation for volume rendering. *IEEE Computer Graphics and Applications*, 20(5):6–6, September/ October 2000.

Lebenslauf — Gunther Heinz Weber

17. Februar 1974	Geburt in Ludwigshafen am Rhein als Sohn von Heinrich August Weber und Ingrid Marie-Luise Weber geb. Müller
12. August 1980	Einschulung in die Seebachschule Osthofen
Sommer 1994	Gauß Gymnasium Worms
15. Juni 1993	Abitur (Leistungskurse: Geschichte — Mathematik — Physik; 4. Prüfungsfach: Deutsch)
Wintersemester 1993/94	Immatrikulation an der Universität Kaiserslautern im Studiengang Informatik mit Nebenfach Physik
12. September 1995	Vordiplom in Informatik
September 1998 – Juni 1999	Forschungsaufenthalt an der University of California, Davis, CA, U.S.A.
14. Mai 1999	Diplom in Informatik
September 1999 – Oktober 2000	Wissenschaftlicher Mitarbeiter am Deutschen Forschungszentrum für Künstliche Intelligenz
Januar 2000 – Juni 2000	Forschungsaufenthalt an der University of California, Davis, CA, U.S.A.
Juli 2000 – Oktober 2000	Forschungsaufenthalt am Lawrence Berkeley National Laboratory
November 2000 – Januar 2001	Wissenschaftlicher Mitarbeiter an der Universität Kaiserslautern (AG Graphsiche Datenverarbeitung und Computergeometrie)
Februar 2001 – Juli 2001	Forschungsaufenthalt an der University of California, Davis, CA, U.S.A.
November 2001	Wissenschaftliche Hilfskraft an der Universität Kaiserslautern (AG Graphsiche Datenverarbeitung und Computergeometrie)
Dezember 2001 – August 2002	Wissenschaftlicher Mitarbeiter an der Universität Kaiserslautern (AG Graphsiche Datenverarbeitung und Computergeometrie)

September 2002 – Oktober 2002	Wissenschaftliche Hilfskraft an der Universität Kaiserslautern (AG Graphsiche Datenverarbeitung und Computergeometrie)
seit November 2002	Forschungsaufenthalt an der University of California, Davis, CA, U.S.A.