Classification and Visualization of Critical Points in 3D Vector Fields

Master thesis
(60 credits)

Martin Aasen
Kristoffer Furuheim

01. May 2008
## Contents

1 Introduction ........................................... 1

2 Background ........................................... 5
  2.1 Vectors ........................................... 5
  2.2 Vector Fields ....................................... 5
    2.2.1 Field Lines .................................... 5
    2.2.2 Critical Points ................................ 7
    2.2.3 Discrete Representation and Interpolation ...... 7
    2.2.4 Numerical Differentiation ....................... 9
    2.2.5 Linear Approximation ........................... 10
  2.3 Eigenvalues and Eigenvectors ....................... 10
    2.3.1 Multiplicity .................................... 10
    2.3.2 Generalized Eigenvectors ....................... 11
  2.4 Phase Planes and Phase Portraits .................... 11
  2.5 Jordan Normal Form and Diagonalization ............ 13

3 Locating First-Order Critical Points in Vector Fields ........... 14
  3.1 Greene’s Bisection Method .......................... 14
    3.1.1 Topological Degree ............................. 15
    3.1.2 Description of the Method ....................... 16
  3.2 Analytic Method for Locating Critical Points ......... 17
    3.2.1 Description of the Method ....................... 18
    3.2.2 Sylvester Matrix and the Resultant Polynomial . 19
    3.2.3 Handling Degenerate Cases ....................... 21
    3.2.4 Numerical Issues ............................... 21
  3.3 Implementation Details .............................. 22
    3.3.1 Reduction Stage ............................... 24
    3.3.2 Newton-Raphson Iteration ....................... 24
    3.3.3 Duplicate Removal ............................... 24
    3.3.4 Validation .................................... 25
  3.4 Comparison and Results ............................. 25
  3.5 Higher-Order Critical Points ....................... 28

4 Classification of First-Order Critical Points in Vector Fields .. 29
  4.1 Motivation ........................................ 29
  4.2 Previous Work ..................................... 29
  4.3 Method ........................................... 30
  4.4 Ordinary Differential Equations ..................... 30
Chapter 1

Introduction

Visualization has over the last decades become a very important part of many scientific areas. Scientists perform more and more accurate observations and simulations, partly because methods and knowledge improve, but also because computational resources increase rapidly, and the experimental techniques improve, allowing the production of larger and more detailed data sets. In computational fluid dynamics (CFD) for example, such a data set is often a numerical representation of a vector field in three dimensions. These data sets are basically just numbers, structured in such a way that when read one by one, or three by three, is like observing the world through a keyhole. An example would be the components of a single vector in a given location in a vector field. The scientist, running a wind tunnel experiment, is then able to see the direction and magnitude of the wind, at a specified point, at a specific time in the tunnel. This is done by looking up the data at the corresponding location in the file. The knowledge of the data in a single point would be of little help when the goal is to obtain knowledge and understanding of the whole flow through the tunnel, or the results of a simulation of a hurricane. The individual numbers can be of little interest. It is the connection between them which is important. The human mind is nowhere capable of scanning through a list of thousands of numbers and obtain an understanding of the overall meaning of a complex data set. The old saying: “a single image says more than a thousand words” summarizes the importance of computer based visualization of scientific data to the point. Visualization is by definition the transformation of data into images [36].

There are many forms of the transformation in question, meaning there are many ways of visualizing the same data. A vector field is essentially a table of vectors, where each vector defines a direction and a magnitude of something, at the position represented by the cell where it resides. This something is typically a fluid flow, making the field a velocity field. Glyphs were among the first means of visualizing this kind of data. A glyph is a geometric object which can depict a direction and magnitude, but in its simplest form it can be a dot marking an interesting position. The arrowhead is a common glyph in 2D vector field visualizations. At a selected set of points, an arrowhead is drawn, and the direction of the arrowhead depicts the direction of the flow in that point, while the length of the arrowhead gives a relative measure of the magnitude of the flow. Now, imagine drawing such an arrowhead in every available point on a screen, i.e. in every pixel. This would lead to a full screen of a single color, due to the fact that every pixel is assigned the color used to encode the arrowheads. There are thousands, maybe millions of arrowheads drawn in this image, but because they are drawn on top of each other, it is impossible to tell. This is a phenomena called clutter, taken to its furthest extent. The size of each arrowhead, and the density of arrowheads are variables which control not only the amount of information encoded in the image, but also the amount of useful information,
which is information the user is able to interpret.

There are techniques that are better suited than glyphs for the visualization of vector fields. One of these techniques is field line visualization. A field line is the path of a test particle advected in space by the vector field at an instant of time. A field line is derived by integrating from a starting point, called a seed point, in the direction of the underlying flow. For every point on the line, the tangent of the line will coincide with the vector in that point. By visualizing a set of field lines, the instant direction of the flow is clearly depicted along the lines, but the magnitude is not encoded unless, for instance, color coding is used. Field lines are often used as a part of the visualization of three dimensional vector fields, and will be explained in detail and used in the visualizations provided in this thesis.

Another well known technique is Line Integral Convolution (LIC) [4]. LIC produces an image where the field lines are given a color intensity so that adjacent lines have different color intensities. There is no clutter problem in the 2D variant of LIC, but the technique is not as effective in three dimensions.

Obtaining a good visualization is challenging, and there will always be a trade-off between the amount of information visualized, and how easily the viewer can interpret this information. The less information to encode, the better are the chances of producing a good visualization. Obviously, making a good choice in what information to visualize is important. Scientist handle this, for example, by calculating different properties of a vector field, and analyzing them separately. Different spatial areas of interest can be defined. These areas, or regions, are often referred to as regions of interest (ROI). ROIs can be areas with a certain value of a specific derived property, visualized with the focus on other properties, to help establish a connection between them. Examples of such properties are pressure and vorticity (which for incompressible flows can be derived from the velocity field).

There are several ways of describing the properties of a vector field. One can separate between quantitative and qualitative properties. Quantitative properties are the properties that can be measured, such as the velocity or pressure at a particular location of a fluid. The qualitative properties are the properties of the vector field that cannot be measured. One such qualitative property of a vector field is its topological structure.

It was Henri Poincaré who laid the mathematical foundation for geometric interpretation of vector fields associated with dynamical systems at the end of the 19th century [13]. This has later been applied by scientist to do qualitative analysis of vector fields, especially in the analysis of large vector datasets from CFD simulations. In these datasets, complex structural behaviors are investigated in the light of their topology. A topological, qualitative analysis of a vector field leads to a separation of the vector field domain into subdomains of similar topological structure [14]. A visualization constructed with this structure in mind yields a representation that describes the fundamental characteristics of a vector field. Such a topology-based visualization is very appealing for large datasets because it reduces the amount of data necessary for interpretation of the visualization. This in turn will increase the probability of having a visualization that is easily interpreted, while still capturing the important features of the vector field.

In this thesis, we will focus on critical points in three-dimensional vector fields, and define regions of interest to be close neighborhoods around these points. Critical points in vector fields are points with zero magnitude. As a real life example, the center of a whirlpool in a river can be such a critical point. These points are important because together with the nearby surrounding vectors, they have more information encoded in them than any such group in the vector field, regarding the total behaviour of the field. Critical points can be regarded as three dimensional traffical intersections, where flows from different directions meet, crash, or deflect each other in directions which can be completely different from their approaching directions.
Introduction

There exist a finite set of fundamentally different critical points, defined by the number of inflow and outflow directions, spiraling structures etc., and combinations of these. Since the set is finite, each critical point can be classified. Such a classification defines the field completely in a close neighborhood around the critical point. By knowing the location and classification of critical points in a vector field, the topology of the field is known in small areas around these. Assuming a smooth transition between these areas, one can construct a simplified model of the whole vector field. Such a simplified representation is useful, for instance, in compressing vector field data into simpler building blocks [30].

Critical points of vector fields are also important in physical sciences. For instance, critical points play an important role in the magnetic field of the Sun, where they are believed to be related to the formation of solar flares [2]. Solar flares are gigantic energy bursts emerging from the surface of the Sun. These energy bursts cause electromagnetic effects that can destroy satellites orbiting the Earth. Another example of an application for critical point analysis is flow simulation of aluminium die casting [8]. There is a relationship between air bubbles in the aluminium die casting, and the critical points of the flow field. Air entrapment in the die casting process causes unwanted porosities, and therefore reduces the material strength of the finished product.

Locating the critical points in a 3D vector field is not a trivial task. They are seldom located exactly on the grid points of the discrete vector field, but at intermediate locations. How the vector fields are defined at these intermediary locations depends on the interpolation method used. Several proposals have been made as how to locate critical points [12, 2, 8, 10], but they have limitations with regards to accuracy and robustness.

The information derived from the classification of critical points aids the information selection process when it comes to visualizing the field. By choosing seed points for field lines based on the topology of critical points, field lines encoding important information is ensured. A more advanced approach is to connect critical points, and use the connecting lines and surfaces to separate areas of different flow topology [16, 18, 43].

We have in this thesis improved an analytic method for locating critical points in 3D vector fields [11]. The improvements are better accuracy, robustness and performance. In addition, we have applied some of the improvements to a previously proposed method by Greene [12]. We test and compare the two improved methods for both analytic vector fields, and velocity field datasets from two large CFD simulations [45]. For classifying critical points, we have constructed a classification scheme based on linear properties of the vector field at the location of a critical point. This scheme is also able to handle degenerate cases. The classification has been derived from the theory of critical points in systems of ordinary differential equations (ODEs) [34]. The topology of solutions to systems of ODEs can be described by the trajectories of the solution of the system of ODEs in phase planes. These phase planes play an important role in our classification scheme, and in the visualization methods we have developed.

To visualize the critical points, and the surrounding vector field topology, we have developed three different visualization methods. These visualization methods are based on the aforementioned classification scheme. In the first method, we visualize the field lines in the vicinity of a critical point. The field lines are placed using a templated strategy that adapts to the classification of critical points. For instance, if a critical point is classified as being part of a spiral vector field structure, such as a vortex, the field lines are placed to capture the spiral topology so that it is clearly visible in the visualization. The second visualization method is a specialized variant of line integral convolution, called Tangent LIC, that captures the tangential directional component of a vector field in planes slicing the vector field. We use these planes to depict the phase planes of critical points. The third visualization method we present, is a hybrid method that combines the two methods into a single visualization.
Organization of the Thesis

This thesis is organized as follows: In chapter 2, we introduce the mathematical definitions used throughout the thesis. Chapter 3 covers strategies for locating critical points in 3D vector fields, followed by chapter 4 which introduces a classification scheme for critical points. Chapter 5 describes three different visualization strategies for critical points, where the third is a combination of the two first. The thesis ends with a conclusion, and some suggestions for extending the presented work.

Acknowledgments

We have received valuable help from a number of people during the work on this thesis. First and foremost, we would like to thank our excellent supervisors at FFI (Norwegian Defence Research Establishment), Øyvind Andreassen and Anders Helgeland, for the help, encouragement and interesting discussions throughout the work on this thesis. In addition, we would like to thank Joseph Werne from Colorado Research Associates Division for providing us with a dataset of a Kelvin-Helmholtz instability simulation [45].
Chapter 2

Background

In this chapter we introduce the mathematical preliminaries for this thesis.

2.1 Vectors

Vectors are mathematical objects that contain information about direction and magnitude. The formal definition of a vector is that it is an element of a vector space. A vector space is a set of vectors that is closed under vector addition and scalar multiplication. For a vector $\mathbf{v} \in \mathbb{R}^n$ in Euclidean $n$-space we use the notation

$$\mathbf{v} = (v_1, v_2, \ldots, v_n),$$

where $v_1, v_2, \ldots, v_n$ are the vector components.

2.2 Vector Fields

Vector fields can be described for general vectors in $\mathbb{R}^n$, but for our purposes it is sufficient to consider $\mathbb{R}^3$, and we restrict the analysis to Cartesian coordinates. A vector field in $\mathbb{R}^3$ is a map that for each point $\mathbf{p} = (x, y, z)$ in a domain assigns a vector

$$\mathbf{F}(\mathbf{p}) = \mathbf{F}(x, y, z) = (F_1(x, y, z), F_2(x, y, z), F_3(x, y, z)), \quad F_i : \mathbb{R}^3 \rightarrow \mathbb{R}.$$  

Each of the functions $F_i$ is a scalar field. The arguments of the function can also include time as a fourth variable if the vector field is time-dependent. In this thesis we will work solely with time-independent vector fields.

Vector fields describe physical quantities such as force, pressure gradients, velocity and others. An example of a 2D vector field is the rotational field shown in figure 2.1. Each arrow indicates the direction and magnitude of the vector located at the starting point of the arrow. This field could represent a rigid-body rotation around the origin, or a vortex in a fluid flow.

2.2.1 Field Lines

A field line is a line that is everywhere tangent to a given vector field at an instant in time. That is, every point on the line has a tangent that coincides with the vector at that location in the vector field. It can be constructed by tracing a path in the direction of the vector field while keeping time fixed. Field lines will depict the direction of the vector field, but not the magnitude.
Let \( \mathbf{r}(s) \) be a field line with parameter \( s \) representing the arc length along the field line. The field line is then given by a system of ordinary differential equations, which written on vector form is

\[
\frac{d\mathbf{r}}{ds} = \mathbf{F}(\mathbf{r}(s)), \quad \mathbf{r}(0) = \mathbf{r}_0, \tag{2.1}
\]

where \( \mathbf{r}_0 \) is an initial point. When numerical integration is used to solve the system of equations for visualization purposes, the point \( \mathbf{r}_0 \) is often referred to as a seed point. In terms of differential equations, the visualization of the field lines of a vector field correspond to phase portraits of solutions of the system (2.1). This will be an important concept in the discussion of classification of critical points in chapter 4.

**Fourth-Order Runge-Kutta Method**

To solve the system of ordinary differential equations (2.1), we need to use a suitable numerical method such as fourth-order Runge-Kutta [5]. Given a point \( \mathbf{r}_0 \) on the field line, the update rule to find the next point along the field line is

\[
\mathbf{r}_{i+1} = \mathbf{r}_i + \frac{h}{6} \left( k_1 + 2k_2 + 2k_3 + k_4 \right),
\]

where \( h \) is the step size, and the intermediate vectors are

\[
\begin{align*}
k_1 &= \mathbf{F}(\mathbf{r}_i) \\
k_2 &= \mathbf{F}(\mathbf{r}_i + \frac{h}{2}k_1) \\
k_3 &= \mathbf{F}(\mathbf{r}_i + \frac{h}{2}k_2) \\
k_4 &= \mathbf{F}(\mathbf{r}_i + hk_3).
\end{align*}
\]

This method can be applied to integrate the field line from a given initial point in the forward direction of the vector field. For backward integration, a sign reversal is required.
2.2.2 Critical Points

A critical point in a vector field is defined as an isolated point $\mathbf{x}$ where the magnitude of the vector field is zero, i.e. $|\mathbf{F}(\mathbf{x})| = 0$. At critical points, the direction of the field line is indeterminate, and they are the only points in the vector field where field lines can intersect (asymptotically). The terms singular point, null point, neutral point or equilibrium point are also frequently used to describe critical points. A critical point is said to be isolated, or simple, if the vector field is non-vanishing in an open neighborhood around the critical point. The requirement that $|\mathbf{F}(\mathbf{x})| = 0$ is identical to $F_i(\mathbf{x}) \cdot F_i(\mathbf{x}) = 0$. For this to be fulfilled, every vector component $F_i$ must be zero at $\mathbf{x}$.

Critical points in vector fields can be classified according to their order. The order of the critical point is identical to the topological degree of the critical point. For a closed surface $\gamma(c)$ surrounding a single critical point $c$ in a continuous vector field, the topological degree of $c$ is defined as the following surface integral [25]:

$$I(c) = \frac{1}{4\pi} \int_{\gamma(c)} d\theta.$$  

Here, $\theta$ is the angle between the vector field and a constant reference vector. The value of $I(c)$ is always an integer, and the order of the critical point is the absolute value of this integer. A surface $\gamma$ enclosing a region which does not contain a critical point has a topological degree of zero.

2.2.3 Discrete Representation and Interpolation

Consider a vector field given on a uniform discrete grid. A uniform discrete grid is a grid where the grid points are evenly spaced in all dimensions. A vector is associated with every grid point. An example in 2D is shown in figure 2.2. In 2D, each square in the grid is referred to as a cell, and each cell has four associated vectors at its vertices. Extending the concept of a cell to three dimensions, the cell is a cube where 8 vectors are specified, one at each vertex of the cube. See figure 2.3 for details and how the indexing order is defined. The line segment between two connected vertices we refer to as an edge, and one side of the cell we call a face. Many of the analysis methods and visualization techniques we describe later use vectors at positions not directly on the grid points of the discrete representation of a vector field. Therefore, it is necessary to use an interpolation scheme to approximate the vectors at intermediate points between grid cells. A suitable interpolation scheme is the trilinear interpolation method. For this interpolation method to give a good approximation, it is assumed that the resolution of the
grid is sufficient, so that the vector field is linear, or nearly linear, within every grid cell. The trilinear interpolant for a cell is

\[ f(x, y, z) = a + bx + cy + dz + exy + f xz + gyz + hxyz. \]  

(2.2)

The variables \( a, b, \ldots, h \) are referred to as interpolation coefficients. Trilinear interpolation of vectors is performed by interpolating each component separately. For a 3D vector field defined using Cartesian coordinates, the set of interpolation formulas is therefore

\[
\begin{align*}
F_1(x, y, z) &= a_1 + b_1 x + c_1 y + d_1 z + e_1 xy + f_1 xz + g_1 yz + h_1 xyz \\
F_2(x, y, z) &= a_2 + b_2 x + c_2 y + d_2 z + e_2 xy + f_2 xz + g_2 yz + h_2 xyz \\
F_3(x, y, z) &= a_3 + b_3 x + c_3 y + d_3 z + e_3 xy + f_3 xz + g_3 yz + h_3 xyz,
\end{align*}
\]

where \( F_1, F_2 \) and \( F_3 \) are the \( x, y \) and \( z \)-components, respectively. To perform an interpolation, the values of the interpolation coefficients must be determined. A system of linear equations where the known values are given by the components of the vectors at the vertices of the cell must be solved. Each set of coefficients, i.e. one set for each vector component, must be solved separately. This system is simplified by introducing a local parametric coordinate system for the cell. In this local coordinate system each component lies in the closed interval \([0, 1]\). This will remove many terms from the system because of the factor zero that is introduced, and we can determine the interpolation coefficients by solving a simpler system of equations:

\[
\begin{align*}
f_0 &= f(0, 0, 0) = a \\
f_1 &= f(1, 0, 0) = a + b \\
f_2 &= f(0, 1, 0) = a + c \\
f_3 &= f(1, 1, 0) = a + b + c + e \\
f_4 &= f(0, 0, 1) = a + d \\
f_5 &= f(1, 0, 1) = a + b + d + f \\
f_6 &= f(0, 1, 1) = a + c + d + g \\
f_7 &= f(1, 1, 1) = a + b + c + d + e + f + g + h.
\end{align*}
\]
Solving this system for the interpolation coefficients gives

\[
\begin{align*}
    a &= f_0 \\
    b &= -f_0 + f_1 \\
    c &= -f_0 + f_2 \\
    d &= -f_0 + f_4 \\
    e &= f_0 - f_1 - f_2 + f_3 \\
    f &= f_0 - f_1 - f_4 + f_5 \\
    g &= f_0 - f_2 - f_4 + f_6 \\
    h &= -f_0 + f_1 + f_2 + f_4 - f_3 - f_5 - f_6 + f_7.
\end{align*}
\]

Interpolated values for the grid cell is calculated by first computing the interpolation coefficients, and then by inserting them into (2.2). This is done on a per-component basis for vectors.

### 2.2.4 Numerical Differentiation

Some of the operations described in this thesis requires the computation of partial derivatives \( \frac{\partial F_i}{\partial x_j} \) at arbitrary points in the vector field. The discrete representation requires interpolation of the derivative as well. The strategy is to first determine the cell that contains the point at which the partial derivative is to be calculated. For this cell, the partial derivative at each of the 8 vertices is computed using a central difference method, and then trilinear interpolation is used to interpolate the derivatives to approximate the partial derivative at a given location within the cell.

Consider the Taylor series expansion around a point \( x \). The forward expansion, assuming \( f \) has continuous derivatives up to second order, is

\[
f(x + h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + O(h^3),
\]

and the corresponding backward expansion is

\[
f(x - h) = f(x) - hf'(x) + \frac{h^2}{2} f''(x) + O(h^3).
\]

By subtracting and rearranging these two equations we get an expression for the derivative:

\[
f'(x) = \frac{f(x + h) - f(x - h)}{2h} + O(h^2).
\]

This is a central-difference scheme for approximating the derivative of \( f \) with truncation error of order \( O(h^2) \). Using the vector-valued function \( F \), the partial derivative can be expressed as

\[
\frac{\partial F_i}{\partial x} = \frac{1}{2h} (F_i(x + h, y, z) - F_i(x - h, y, z)).
\]

By letting \( h \) be the length of the side of a grid cell, this equation approximates the partial derivative of every component \( F_i \) of \( F \) with respect to \( x \). Similar formulas can be used for computing the partial derivative with respect to \( y \) and \( z \). Given this formula the partial derivative at each of the vertices of a grid cell can be computed. Trilinear interpolation is used to approximate the partial derivative at an arbitrary point within the cell by interpolating the values of the partial derivatives computed at the vertices.

This formula cannot be used for the cells on the boundary of the vector field grid. For these cells, the central-difference is replaced by a simple forward- or backward-difference, depending on which boundary of the vector field the cell is located at.
2.2.5 Linear Approximation

For some of the methods described later in this thesis it is necessary to use a linear expansion of the vector field around a given point. A linear expansion gives us information that will be used to classify critical points in chapter 4. This information is deduced from the Jacobian matrix \( \frac{\partial F_i}{\partial x_j} \), containing all first-order partial derivatives, evaluated at the critical point.

Suppose that the points \( p = (x_i) \) and \( p_0 = (x_{i0}) \) are close to each other, and that all first-order partial derivatives are continuous at \( x_{i0} \). The value at \( x_i \) can be approximated using a multivariate Taylor series expansion. For each component \( F_i \), there is a first-order approximation formula

\[
F_i(x_j) \approx F_i(x_{j0}) + \frac{\partial F_i(x_{j0})}{\partial x_k}(x_k - x_{k0}),
\]

(2.3)

where the Einstein summation convention for repeated indices has been used. Second-order terms, and higher, have been truncated. In the case of \( \mathbf{F} \) representing a velocity, the Jacobian matrix is referred to as the velocity gradient matrix, or the rate-of-deformation tensor. If \( x_{i0} \) is a critical point, the formula simplifies to

\[
F_i(x_j) \approx \frac{\partial F_i(x_{j0})}{\partial x_k}(x_k - x_{k0}).
\]

2.3 Eigenvalues and Eigenvectors

The classification of a critical point depends on the eigenvalues of the Jacobian matrix evaluated at the critical point. Let \( A \) be an \( n \times n \) matrix, and \( \mathbf{v} \) a vector of \( n \) components. An eigenvalue of \( A \) is a scalar \( \lambda \) satisfying the equation [24]:

\[
Av = \lambda \mathbf{v}.
\]

(2.4)

This is identical to the requirement that the equation

\[
(A - \lambda I)\mathbf{v} = \mathbf{0}
\]

has nontrivial solutions for \( \mathbf{v} \). The set of solutions form a subspace of \( \mathbb{R}^n \) known as the eigenspace of \( A \) corresponding to the eigenvalue \( \lambda \). For square matrices there exists a scalar equation

\[
\det(A - \lambda I) = 0
\]

(2.5)

known as the characteristic equation. The left side of this equation is a polynomial of degree \( n \), and the eigenvalues of \( A \) are the \( n \) roots of this polynomial.

If \( A \) is not only a square matrix, but also a triangular matrix, the eigenvalues are given by the entries on its main diagonal. This triangular matrix clearly displays possible eigenvalues, while the off-diagonal entries determine the number of free variables, which again determines the degree of freedom of the eigenvector’s direction. The number of free variables will affect the number of linearly independent eigenvectors when two or more of the eigenvalues are equal.

We will use the term eigenpair to refer to the pairs \((\lambda_i, \mathbf{v}_i)\) of eigenvalues and eigenvectors, and the term eigenset to refer to all \( n \) eigenpairs of a given matrix.

2.3.1 Multiplicity

The multiplicity of the eigenvalues of a matrix \( A \) is separated into algebraic and geometric multiplicity [24]. In this thesis, both terms will be used extensively in the classification of critical points presented in chapter 4.
The algebraic multiplicity of an eigenvalue is the number of times this exact value occurs as an eigenvalue of the matrix. If an eigenvalue \( \lambda_0 \) has algebraic multiplicity \( m_a \), \( m_a \) roots of the characteristic equation \( (A - \lambda I) = 0 \) equals \( \lambda_0 \). If \( m_a = 1 \), \( \lambda_0 \) is called simple, if \( m_a > 1 \), \( \lambda_0 \) is called repeated. The total algebraic multiplicity of a matrix is the sum of the arithmetic multiplicity of each of its eigenvalues, and is always equal to \( n \) for an \( n \times n \) matrix.

The set of linearly independent eigenvectors corresponding to an eigenvalue with algebraic multiplicity \( m_a \), can be comprised of one eigenvector, or up to \( m_a \) eigenvectors.

The geometric multiplicity \( m_g \) of an eigenvalue is the size of this set, and equals the dimensions of the eigenspace of the eigenvalue. This is equivalent to the dimensions of the null space of the matrix \((A - \lambda I)\), which again equals the number of free variables of the equation \((A - \lambda I) = 0\).

Note that when \( m_g \) is larger than one, so is the dimensions of the eigenspace of the corresponding eigenvalue, and all vectors in this eigenspace are eigenvectors of the matrix \( A \). Thus, when an eigenvalue has a geometric multiplicity larger than one, the direction of the corresponding eigenvectors are not uniquely determined.

### 2.3.2 Generalized Eigenvectors

A matrix \( A \) is called a defective matrix when it does not have a full set of eigenvectors. That is, when it has fewer linearly independent eigenvectors than it has eigenvalues. Generalized eigenvectors are used to form a basis of such a matrix. A generalized eigenvector fulfills the equation

\[
(A - \lambda I)^k \mathbf{v} = 0, \tag{2.6}
\]

when \( \lambda \) has multiplicity \( k \). It can also be computed recursively by solving the equation

\[
(A - \lambda I) \mathbf{v}_i = \mathbf{v}_{i-1} \tag{2.7}
\]

for \( i = 1, \ldots, k \) and \( \mathbf{v}_0 = 0 \). This results in \( \mathbf{v}_1 \) being the real eigenvector, and the following vectors being generalized eigenvectors.

### 2.4 Phase Planes and Phase Portraits

Phase planes play an important role in the classification of critical points. A phase plane is a term used in the theory of differential equations, and refers to a plane containing solution trajectories of a system of differential equations [34]. The infinite set of solution trajectories in this plane is called a phase portrait. In this thesis, the terms will be used in the classification of critical points.

Consider a plane spanned by two linearly independent vectors \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \) as shown in figure 2.4. Then define two functions \( x(t) \) and \( y(t) \) giving a position \((x, y)\) in the plane, in terms of the unit vectors, at the time \( t \)

\[
x(t) = c_1 e^{at},
y(t) = c_2 e^{bt}.
\]

These functions can be collected in a vector function in the following way

\[
x(t) = c_1 \mathbf{v}_1 e^{at} + c_2 \mathbf{v}_2 e^{bt}. \tag{2.8}
\]
Figure 2.4: A phase plane with a single solution trajectory drawn. The point \((x_0, y_0)\) defines the initial condition.

The constants \(c_1\) and \(c_2\) define the position \(x(t)\) at \(t = 0\), and is chosen to satisfy an initial condition \((x_0, y_0)\)

\[
\begin{align*}
x(0) &= c_1e^{a\cdot0} = c_1 = x_0 \\
y(0) &= c_2e^{b\cdot0} = c_2 = y_0,
\end{align*}
\]

or

\[
x(0) = c_1v_1e^{a\cdot0} + c_2v_2e^{b\cdot0} = c_1v_1 + c_2v_2 = v_c.
\]

When the constants \(c_1\) and \(c_2\) which meet the initial condition have been determined, an initial point \(x_0v_1 + y_0v_2\), and the curve passing through it, is defined. The curve is made up of all the points given by \(x(t)\) evaluated for parameter \(t \in \mathbb{R}\).

In chapter 4, the theory from this section will be used to classify critical points in three-dimensional vector fields. The function \(x(t)\) will then be a part of a solution of a set of differential equations. Each curve in the phase plane will be a solution trajectory of this set, and a first-order approximation of a field line of the vector field the differential equations are derived from.

When the lines are solution trajectories or field lines, the orientation of the vector \(x(t_1) - x(t_0)\) where \(t_1 > t_0\) becomes important. The sign of the exponential factors \(a\) and \(b\) in equation (2.8) determine the direction of flow in direction of \(v_1\) and \(v_2\), respectively. If \(a\) is negative and \(b\) is positive, a point \(p\) given by \(x(t)\) will approach the origin in the direction of \(v_1\) as \(t\) increases, and then deflect and move away from the origin in the direction of \(v_2\). Such a set of curves is called a saddle and is shown in figure 4.1.
2.5 Jordan Normal Form and Diagonalization

The Jordan normal form of a matrix unveils the algebraic and geometric multiplicities of the eigenvalues of the matrix, and is therefore important in the classification of critical points. An \( n \times n \) matrix can be written in the form
\[
A = PDP^{-1},
\]
where \( D \) is a diagonal matrix with the eigenvalues of \( A \) on its main diagonal. The columns of \( P \) are the corresponding eigenvectors of \( A \), appearing in the same order as the eigenvalues. If \( A \) is a defective matrix, which means that \( A \) does not have \( n \) linearly independent eigenvectors, a different approach is needed to diagonalize \( A \). Instead of using the diagonal matrix \( D \), the Jordan normal form \( J \) of matrix \( A \) is used, and one or more of the columns of \( P \) are generalized eigenvectors. The Jordan normal form \( J \) is constructed from smaller matrices called Jordan blocks. Consider an eigenvalue \( \lambda_i \) of the \( A \) with algebraic multiplicity \( m_a \) and geometric multiplicity \( m_g \). From this eigenvalue, \( m_g \) Jordan block matrices are constructed. Each Jordan block matrix \( B_k \) has the form
\[
B_k = \begin{bmatrix}
\lambda_i & 1 & \cdots & \cdots & 1 \\
\lambda_i & \ddots & \ddots & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots \\
\ddots & \ddots & \ddots & \ddots & \ddots \\
1 & \ddots & \ddots & \ddots & \lambda_i \\
\end{bmatrix},
\]
where \( B_k \) denotes the \( k \)-th Jordan block matrix of an eigenvalue. The contribution from an eigenvalue, in the form of Jordan blocks, is \( m_a \) blocks. If \( \lambda_i \) has a geometric multiplicity equal to the algebraic multiplicity, the contribution is \( m_a \) blocks of dimension \( 1 \times 1 \). These are simply the eigenvalue \( \lambda_i \). If \( m_a > m_g \), the last Jordan block matrix has \( (m_a - m_g) + 1 \) rows and columns, making the total dimensions of the Jordan blocks corresponding to \( \lambda_i \), \( m_a \times m_a \). This last block will have one entry on the upper diagonal for each generalized eigenvector that needs to be used to make the set of eigenvectors linearly independent. Thus, if an eigenvalue has a complete set of linearly independent eigenvectors, i.e. the geometric multiplicity equals the algebraic, there is no entry on the upper diagonal of the collected block matrix contributed by this eigenvalue.

If the matrix \( A \) is defined to be
\[
A = \begin{bmatrix}
5 & 4 & 2 & 1 \\
0 & 1 & -1 & -1 \\
-1 & -1 & 3 & 0 \\
1 & 1 & -1 & 2 \\
\end{bmatrix},
\]
the eigenvalues are 1, 2, 4, 4. The eigenvalue 4 is repeated, and the set of eigenvectors corresponding to it has only one linearly independent vector. The corresponding Jordan normal form is then
\[
J_n = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 4 & 1 \\
0 & 0 & 0 & 4 \\
\end{bmatrix},
\]
Since the eigenvalues 1 and 2 are not repeated, \( m_a = m_g \) for both, and the contribution is one \( 1 \times 1 \) Jordan block \([1]\), corresponding to the eigenvalue 1, and one \( 1 \times 1 \) Jordan block \([2]\), corresponding to the eigenvalue 2. The eigenvalue 4 has \( m_a = 2 \) and \( m_g = 1 \), making the contribution from this eigenvalue a single Jordan block \([4 \ 1 \ 0 \ 4]\).
Chapter 3

Locating First-Order Critical Points in Vector Fields

Several approaches have been proposed for locating first-order critical points in 3D vector fields. Among the first was Greene’s bisection method [12]. This method is based on the topological degree of a critical point as defined in section 2.2.2. Topological degree theory cannot be used to reveal the exact location of a critical point directly, but is used for determining if a given region of the vector field can possibly contain a critical point. By bisecting regions which have been determined to contain at least one critical point, Greene’s method proceeds by recursively analyzing smaller and smaller regions that are known to contain at least one critical point until a size criteria for the regions are met. The location of the critical point within this region can then be approximated.

A different approach is presented in [8]. This approach divides the vector field into tetrahedral cells, and uses linear interpolation within each tetrahedral cell to try to locate critical points. Because of this partitioning of hexahedral cells into tetrahedral cells, multiple critical points can be located inside a hexahedral grid cell.

To help reduce the computational time of the critical point locating process, a parallel method suitable for multiprocessor systems was introduced in [10]. The algorithm used for locating critical points is similar to the algorithm in [12]. The results from this work show that multi-core processor architectures are advantageous with regards to the extraction of critical points in large datasets.

A new method that takes a direct approach by solving the trilinear interpolation equations for critical points, was presented in a work by Gjøystdal [11]. This method should be able to locate all first-order critical points within a hexahedral grid cell of a discrete vector field.

The reason for limiting our study to first-order critical points is the use of trilinear interpolation, which will reduce the complexity of the topology of the field, thus limiting the type of critical points that can be detected. See section 3.5 for a discussion of higher-order critical points. In the following we present Greene’s bisection method, and the analytic method presented in [11], with improvements in efficiency and robustness.

3.1 Greene’s Bisection Method

Greene’s bisection method, or simply Greene’s method, is inspired by the bisection method for finding roots of equations. Suppose we have a function $f(x)$ that is continuous on an interval $[a, b]$. Given that $f(a)$ and $f(b)$ have opposite signs, we know by the intermediate value theorem that $f$ must have at least one root in the interval $(a, b)$. In each iteration of the bisection
method, the sign of the value at the midpoint \( c = (a + b)/2 \) is determined. If \( f(a) \) and \( f(c) \) have opposite signs, the interval \((a, c)\) must contain one or more roots. Similarly, if \( f(b) \) and \( f(c) \) have opposite signs, the interval \((c, b)\) must contain the root. This interval-testing technique is applied recursively on the sub-intervals where a sign change occurs. The process is continued until sufficient accuracy has been achieved, which can be determined by the size of the sub-interval.

A similar concept to sign change in the root-finding bisection method is the analysis of topological degree. Analysis of the topological degree cannot directly tell the location and type of critical points, but can be used to determine the existence of critical points. The method is used in a recursive fashion similar to that of the bisection method for roots of equations. First, the topological degree is used to determine if a cell can contain a critical point. If so, the cell is subdivided into 8 subcells, and the method is applied to each of these subcells. This process is repeated until the subcell is small enough to be used to determine the location of the critical point with sufficient accuracy.

### 3.1.1 Topological Degree

The term topological degree refers to the Poincaré-index [12] of a critical point. Given a curve immersed in a vector field, the Poincaré-index is defined as the total rotation of the vectors of the vector field along the curve. For a closed curve in a continuous 2D vector field the number of rotations will always be an integer since the start and end vectors will be identical. See figure 3.1 for an example in 2D for an open curve. In 3D, the Poincaré-index is evaluated by a surface integral of the vector field over a closed surface \( \gamma \) (see section 2.2.2). The topological degree of a volume containing a single critical point is given by the sign of the determinant of the Jacobian matrix. Positive determinants result in a topological degree of +1, and negative determinants result in

![Figure 3.1](image-url)

Figure 3.1: Calculating the Poincaré-index. The angles between the vectors are accumulated integral of the vector field over a closed surface \( \gamma \) (see section 2.2.2). The topological degree of a critical point is dependent on the linear behavior of the field in the vicinity of the critical point. Given that a critical point is located at \( x_i = x_{i0} \), the field in the vicinity of the critical point can be approximated by

\[
F_i \approx J_{ij}(x_j - x_{j0}),
\]

where

\[
J_{ij} = \left. \frac{\partial F_i}{\partial x_j} \right|_{x_j = x_{j0}}
\]

is the Jacobian matrix evaluated at the critical point. The topological degree \( D \) of a volume containing a single critical point is given by the sign of the determinant of the Jacobian matrix. Positive determinants result in a topological degree of +1, and negative determinants result in
a topological degree of $-1$. Topological degree is additive. So, for a volume that has multiple critical points, the topological degree of each critical point is added to yield the topological degree for the volume. In general, for a domain $D$ of a given vector field $\mathbf{F}$, the topological degree is given by

$$\deg(\mathbf{F}, D) = \sum_{\sigma} \text{sgn} \left[ \det \left( \frac{\partial F_i}{\partial x_j} \right|_{x_j = x_{j,0}} \right), \quad \sigma = \{ x_j \in D \mid F_i(x_j) = 0 \}.$$ 

Thus, if the volume contains two critical points, they may cancel each other, resulting in a topological degree of 0.

### 3.1.2 Description of the Method

The algorithm for locating a critical point using Greene’s bisection method starts by checking for existence of a critical point within the cell. Each of the six rectangular faces that make up the boundary of the cell are divided into two triangles by connecting the two vertices at opposite ends along one of the the diagonals for the rectangle. There are two choices for this triangulation for each face, so the total number of possible triangulations is $2^6 = 64$. After the triangulation process, the solid angle of each triangle is calculated. For each triangle the vectors $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ are ordered in a right-handed manner about the normal vector pointing outwards from the volume enclosed by the triangles. Together, the 12 triangles represent a dodecahedron with triangular faces. For each of these triangles, an area element is computed. This area element is the solid angle subtended by the three vectors at the vertices of the triangle projected onto the unit sphere. See figure 3.2 for an illustration. Each projection results in a solid angle $A_i$, which is calculated using the formula

$$\tan^2 \left( \frac{A_i}{4} \right) = \tan \left[ \frac{\theta_1 + \theta_2 + \theta_3}{4} \right] \times \tan \left[ \frac{\theta_1 + \theta_2 - \theta_3}{4} \right] \times \tan \left[ \frac{\theta_2 + \theta_3 - \theta_1}{4} \right] \times \tan \left[ \frac{\theta_3 + \theta_1 - \theta_2}{4} \right],$$

where the angles $\theta_1, \theta_2$ and $\theta_3$ can be computed using

$$\cos(\theta_1) = \frac{\mathbf{B}_2 \cdot \mathbf{B}_3}{||\mathbf{B}_2|| ||\mathbf{B}_3||}, \quad \cos(\theta_2) = \frac{\mathbf{B}_1 \cdot \mathbf{B}_3}{||\mathbf{B}_1|| ||\mathbf{B}_3||}, \quad \cos(\theta_3) = \frac{\mathbf{B}_1 \cdot \mathbf{B}_2}{||\mathbf{B}_1|| ||\mathbf{B}_2||}.$$ 

Let $A_i$ be the solid angle of the $i$-th triangle. The contribution $A_i$ is chosen to be negative if the scalar triple product $\mathbf{B}_1 \cdot (\mathbf{B}_2 \times \mathbf{B}_3)$ is negative. Finally, the topological degree is computed
by adding the solid angle from all 12 triangles, and dividing by the solid angle of a sphere:

$$D = \frac{1}{4\pi} \sum_{i=1}^{12} A_i.$$ 

When this process has been completed, further processing is required to determine the actual location of the zero. If the cell has been determined to contain a critical point, the cell is subdivided into 8 subcells. The topological degree for each of these subcells is computed. For the cells that has a nonzero topological degree the process is repeated. This recursive process continues until the subcell is small enough to give the required accuracy.

A parameter to be specified for the bisection method is the maximum number of subdivisions. The value of this parameter will affect the number of critical points located, and there is no way of determining a priori how many subdivisions should be performed to prevent two or more critical points from being in the same subcell, thus potentially failing to locate any of them because of the additive property of topological degree. This value is the depth of the recursion, and we define a depth of 0 to be the cell before any subdivision is performed.

### 3.2 Analytic Method for Locating Critical Points

In [11], a new analytic method for locating critical points is described that should improve upon the topological method described in the previous section. What we present here is the analytic critical point locating method as given in [11], with improvements. We begin by looking at the set of interpolation formulas for trilinear vector fields as defined in section 2.2.3:

$$F_1(x, y, z) = a_1 + b_1x + c_1y + d_1z + e_1xy + f_1xz + g_1yz + h_1xyz \quad (3.1)$$
$$F_2(x, y, z) = a_2 + b_2x + c_2y + d_2z + e_2xy + f_2xz + g_2yz + h_2xyz \quad (3.2)$$
$$F_3(x, y, z) = a_3 + b_3x + c_3y + d_3z + e_3xy + f_3xz + g_3yz + h_3xyz. \quad (3.3)$$

At a critical point, these three equations should all be zero, giving the following system of equations:

$$F_1(x, y, z) = 0$$
$$F_2(x, y, z) = 0$$
$$F_3(x, y, z) = 0$$

This system of equations can be solved using an analytic method. The strategy is as follows: First the interpolation coefficients must be computed. Variable $z$ is then eliminated from one of the equations (3.1) - (3.3). The expression for $z$ is inserted into the two other equations giving two functions $\xi(x, y)$ and $\phi(x, y)$. These two are solved so that $\xi(x, y) = \phi(x, y) = 0$. Each pair $(x, y)$ of solutions of $\xi(x, y)$ and $\phi(x, y)$ is inserted into the expression for $z$, giving a tuple $(x, y, z)$ that is a candidate for a critical point. To confirm that it is an actual critical point, the magnitude of the interpolated vector at location $(x, y, z)$ must be checked to see that it is in fact zero. Since we are using finite-precision floating-point operations, the interpolated vector magnitude might not be exactly zero, so it is necessary to specify a tolerance level for which points are to be considered true critical points. This tolerance level will depend on the type of data being analyzed, and it will affect the number of critical points located.

It is important to note that the analytic critical point method requires that the vector field is sufficiently smooth for a linear subcell interpolation to be valid.
3.2.1 Description of the Method

This section describes how the system of trilinear equations (3.1) - (3.3) can be solved using analytic means. We start by solving equation (3.1) for $z$:

$$z = \frac{a_1 + b_1 x + c_1 y + e_1 xy}{d_1 + f_1 x + g_1 y + h_1 xy}$$  \hspace{1cm} (3.4)

This transformation is substituted into equations (3.2) and (3.3) for $z$. After rearranging and ordering, we get

$$\xi(x, y) = (A_1 x^2 + B_1 x + C_1) y^2 + (A_2 x^2 + B_2 x + C_2) y + (A_3 x^2 + B_3 x + C_3) = 0$$  \hspace{1cm} (3.5)

$$\phi(x, y) = (A_4 x^2 + B_4 x + C_4) y^2 + (A_5 x^2 + B_5 x + C_5) y + (A_6 x^2 + B_6 x + C_6) = 0,$$  \hspace{1cm} (3.6)

where

$$
\begin{align*}
A_1 &= e_2 h_1 - h_2 e_1 \\
B_1 &= e_2 g_1 - g_2 e_1 + c_2 h_1 - h_2 c_1 \\
C_1 &= c_2 g_1 - g_2 c_1 \\
A_2 &= e_2 f_1 - f_2 e_1 + b_2 h_1 - h_2 b_1 \\
B_2 &= e_2 d_1 - d_2 e_1 + c_2 f_1 - f_2 c_1 + b_2 g_1 - g_2 b_1 + a_2 h_1 - h_2 a_1 \\
C_2 &= c_2 d_1 - d_2 c_1 + a_2 g_1 - g_2 a_1 \\
A_3 &= b_2 f_1 - f_2 b_1 \\
B_3 &= b_2 d_1 - d_2 b_1 + a_2 f_1 - f_2 a_1 \\
C_3 &= a_2 d_1 - d_2 a_1,
\end{align*}
$$

and

$$
\begin{align*}
A_4 &= e_3 h_1 - h_3 e_1 \\
B_4 &= e_3 g_1 - g_3 e_1 + c_3 h_1 - h_3 c_1 \\
C_4 &= c_3 g_1 - g_3 c_1 \\
A_5 &= e_3 f_1 - f_3 e_1 + b_3 h_1 - h_3 b_1 \\
B_5 &= e_3 d_1 - d_3 e_1 + c_3 f_1 - f_3 c_1 + b_3 g_1 - g_3 b_1 + a_3 h_1 - h_3 a_1 \\
C_5 &= c_3 d_1 - d_3 c_1 + a_3 g_1 - g_3 a_1 \\
A_6 &= b_3 f_1 - f_3 b_1 \\
B_6 &= b_3 d_1 - d_3 b_1 + a_3 f_1 - f_3 a_1 \\
C_6 &= a_3 d_1 - d_3 a_1.
\end{align*}
$$

For the system (3.1) - (3.3) to have a solution, we must have that

$$
\begin{align*}
\xi(x, y) &= 0 \\
\phi(x, y) &= 0.
\end{align*}
$$

For a given $x$ there must be a common value $y$ that solves both equations. To determine this common factor, if any, one possibility is to use the determinant of the Sylvester matrix described in the section below to find a polynomial which can be solved for $x$. 


3.2.2 Sylvester Matrix and the Resultant Polynomial

Given two polynomials:

\[ p(x) = p_m x^m + p_{m-1} x^{m-1} + \ldots + p_1 x + p_0 \]

of degree \( m \) with roots \( \alpha_i, \ i = 1, \ldots, m \) and

\[ q(x) = q_n x^n + q_{n-1} x^{n-1} + \ldots + q_1 x + q_0 \]

of degree \( n \) with roots \( \beta_j, \ j = 1, \ldots, n \). The resultant \([39]\) is then defined as

\[ R(p, q) = p_m q_n \prod_{i=1}^{m} \prod_{j=1}^{n} (\alpha_i - \beta_j). \tag{3.7} \]

The importance of this result is that it can be used to eliminate one variable from a system of two equations. We observe that if the two polynomials \( p(x) \) and \( q(x) \) have a common root, one of the factors in the product will be zero, and the resultant \( R(p, q) \) will be zero.

The resultant is also given as the determinant of the Sylvester matrix \( S_{p,q} \). Let the two polynomials \( p(x) \) and \( q(x) \) be given as above. The dimensions of the Sylvester matrix is then \((m + n) \times (m + n)\). It is formed by first filling the first row with the coefficients of \( p(x) \), then shifting right one column and down one row and filling in the coefficients until they hit the right side of the matrix. This process is repeated for \( n \) rows. The process is similar for \( q(x) \), whose coefficients begin at one row below the coefficients of \( p(x) \). We look at an example with \( m = 3 \) and \( n = 2 \). The system of polynomial equations is

\[ \begin{align*}
p(x) &= p_3 x^3 + p_2 x^2 + p_1 x + p_0 = 0 \\
q(x) &= q_2 x^2 + q_1 x + q_0 = 0.
\end{align*} \]

These two equations can be written in matrix form as

\[ \begin{bmatrix}
p_3 & p_2 & p_1 & p_0 \\
0 & q_2 & q_1 & q_0
\end{bmatrix} \begin{bmatrix}
x^3 \\
x^2 \\
x \\
1
\end{bmatrix} = \begin{bmatrix}
0 \\
0
\end{bmatrix}. \]

Determinants can only be computed for square matrices. We can extend the matrix by adding more rows to make it a square matrix. To do this we add new equations to the system of equations of the form \( x^k p(x) \) and \( x^k q(x) \). This does not alter the solutions because the extended system of equations will have exactly the same solutions as the original system. We determine the extended system to be

\[ \begin{align*}
xp(x) &= 0 \\
p(x) &= 0 \\
x^2 q(x) &= 0 \\
xq(x) &= 0 \\
q(x) &= 0,
\end{align*} \]

which in matrix form is written as

\[ \begin{bmatrix}
p_3 & p_2 & p_1 & p_0 & 0 \\
0 & p_3 & p_2 & p_1 & p_0 \\
q_2 & q_1 & q_0 & 0 & 0 \\
0 & q_2 & q_1 & q_0 & 0 \\
0 & 0 & q_2 & q_1 & q_0
\end{bmatrix} \begin{bmatrix}
x^4 \\
x^3 \\
x^2 \\
x \\
1
\end{bmatrix} = \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}, \]
where the matrix on the left side is the Sylvester matrix. The Sylvester matrix for \( \xi(x, y) \) and \( \phi(x, y) \) is

\[
S_{\xi,\phi}^1 = \begin{bmatrix}
A_1x^2 + B_1x + C_1 & A_2x^2 + B_2x + C_2 & A_3x^2 + B_3x + C_3 & 0 \\
0 & A_1x^2 + B_1x + C_1 & A_2x^2 + B_2x + C_2 & A_3x^2 + B_3x + C_3 \\
A_4x^2 + B_4x + C_4 & A_5x^2 + B_5x + C_5 & A_6x^2 + B_6x + C_6 & 0 \\
0 & A_4x^2 + B_4x + C_4 & A_5x^2 + B_5x + C_5 & A_6x^2 + B_6x + C_6
\end{bmatrix}.
\]

Note here that the superscript in \( S_{\xi,\phi}^i \) is an index, not an exponent. Computing the determinant of this matrix yields a resultant polynomial \( R^1(x) = \det(S_{\xi,\phi}^1) \) of degree 8.

We need to take special consideration in case either \( \xi(x, y) \) or \( \phi(x, y) \) has one or more leading zero coefficients. For instance, if the \( x \)- and \( y \)-components are null everywhere on the face defined by vertices \( f_0, f_1, f_2 \) and \( f_3 \), then \( \phi(x, y) \) is reduced to a first-order polynomial since \( A_1 = B_1 = C_1 = 0 \). In that case, the Sylvester matrix must be adjusted to compensate for the lower degree polynomial. This was not addressed by Gjøystdal. The additional matrices needed for these degenerate cases are given below.

**Case:** \( A_1x^2 + B_1x + C_1 = 0 \)

\[
S_{\xi,\phi}^2 = \begin{bmatrix}
A_2x^2 + B_2x + C_2 & A_3x^2 + B_3x + C_3 & 0 \\
0 & A_2x^2 + B_2x + C_2 & A_3x^2 + B_3x + C_3 \\
A_4x^2 + B_4x + C_4 & A_5x^2 + B_5x + C_5 & A_6x^2 + B_6x + C_6
\end{bmatrix}.
\]

**Case:** \( A_4x^2 + B_4x + C_4 = 0 \)

\[
S_{\xi,\phi}^3 = \begin{bmatrix}
A_1x^2 + B_1x + C_1 & A_2x^2 + B_2x + C_2 & A_3x^2 + B_3x + C_3 & 0 \\
A_5x^2 + B_5x + C_5 & A_6x^2 + B_6x + C_6 & 0 \\
0 & A_5x^2 + B_5x + C_5 & A_6x^2 + B_6x + C_6
\end{bmatrix}.
\]

**Case:** \( A_1x^2 + B_1x + C_1 = 0 \) and \( A_4x^2 + B_4x + C_4 = 0 \)

\[
S_{\xi,\phi}^4 = \begin{bmatrix}
A_2x^2 + B_2x + C_2 & A_3x^2 + B_3x + C_3 \\
A_5x^2 + B_5x + C_5 & A_6x^2 + B_6x + C_6
\end{bmatrix}.
\]

The determinant of the matrices \( S_{\xi,\phi}^i, i = 1, \ldots, 4 \) can be computed using a mathematical software package that supports symbolic computation. We used Maple for this computation. The resultant polynomial \( R^1(\xi, \phi) \) for \( S_{\xi,\phi}^1 \) is of degree 8 and has 452 terms.

To find the roots of the resultant polynomial, a suitable root-finding numerical method must be used. We used GSL \(^1\), which is a numerical computations library that provides a function for solving for roots of high-order real polynomials. The function uses an iterative balanced QR-reduction of the companion matrix of the polynomial to compute an approximate solution [24]. It should be noted that there is no guarantee that this method will converge. If the method does not converge, we use a multidimensional Newton-Raphson iteration initiated from the center of the cell to try to locate at least a single critical point. This method is also provided by GSL.

When the roots of the polynomial have been determined using the numerical method, the next step is to determine which roots are the candidates for the \( x \)-component of the critical point(s). The candidates are the values that are real and inside the closed unit interval. The candidate values of \( x \) are inserted into \( \xi(x, y) \) and \( \phi(x, y) \) to see if there is at least one common root for \( y \). For each pair \( (x, y) \) that solves both equations, the values are inserted in (3.4) to find the corresponding \( z \)-value.

\(^1\)GNU Scientific Library. Available at: http://www.gnu.org/software/gsl/
Not all tuples \((x, y, z)\) that are computed correspond to true critical points in the cell. For instance, not all \(x\)-values that were marked candidates will correspond to solutions of the trilinear equations. Also, that \(x\) and \(y\) satisfy \(\xi(x, y)\) and \(\phi(x, y)\) implies that \(F_1\) and \(F_2\) have a shared solution for \(z\), but not that all three of the trilinear interpolants \(F_1\), \(F_2\) and \(F_3\) are zero. Every candidate tuple is therefore forwarded to a validation step to test if it is a true isolated critical point.

### 3.2.3 Handling Degenerate Cases

There are multiple examples which make the original analytic critical point locator fail. For instance, the denominator used in the calculation of \(z\) in (3.4) can become zero if the coefficients \(d, f, g\) and \(h\) are all zero. This happens if all vectors at the vertices of the cell are null-vectors. Null-vectors can be common in numerical datasets. For instance, in a CFD (computational fluid dynamics) simulation there can be rigid objects occupying parts of the volume, and the velocity vector will be null in such regions. Another example is null-vectors resulting from boundary conditions in CFD simulations, such as no-slip boundaries, where the velocity of the fluid is zero. We reject these cells from further analysis.

We use a simple mask to avoid testing for critical points in regions where the vector field is known to be zero. The mask is simply a boolean mask filter with one boolean value for each cell. Only those cells with the filter value set to true will be considered when locating critical points. This will ensure no false critical points are returned, and it gives us the possibility of limiting the region for which the search for critical points is performed. Also, for large fields with a large number of critical points, the filter can be used to reduce clutter in a visualization by masking so that only predefined regions are analyzed for critical points.

### 3.2.4 Numerical Issues

The resultant polynomial for \(S^1\xi\phi\) has more than 450 terms divided amongst the 9 coefficients to be computed. Care must be taken when doing this computation since it is sensitive to floating-point cancellation errors. These types of errors occur when performing addition and subtraction of floating-point numbers that are almost equal in absolute value. The result is that the computed values have fewer significant digits than either of the two operands, and even if the absolute error is small, the relative error can be large. Consider the following example:

\[
x = 0.34561234 - 0.3456 = 0.00001234
\]

This calculation gives the real answer when computing by hand. Now suppose we perform the same computation using a machine that has 5 digits of accuracy. The calculation is then

\[
\bar{x} = 0.34561 - 0.34560 = 0.00001
\]

Whereas the real answer \(x\) has 4 significant digits, the computed answer \(\bar{x}\) only has 1 significant digit. The relative error is approximately 0.2, while the absolute error is \(2.34 \cdot 10^{-6}\). These small errors will propagate and cause the total sum to be incorrect.

Our initial testing of the analytic method revealed that for a synthetic vector field generated from analytic expressions, for each vector field component the cancellation errors can cause the initial step of calculating the 8 (possibly complex) roots of the polynomial to fail. This is due to the coefficients of the resultant polynomial being incorrect. This in turn will make the next step of calculating the \(y\)-values fail, hence no value for \(z\) can be computed. To handle this problem, a simple solution is to use arbitrary precision arithmetics for the computation of the coefficients.
However, the use of arbitrary precision numbers comes with a significant performance hit. An alternative approach is to use a different ordering of the terms in the computation for each coefficient. Each term is appended to a list, one list for positive values and one list for negative values. The lists are sorted and the values are added from the lowest and up. This gives one sum for each list, and these two values are subtracted to give the coefficient. The cancellation error should be reduced when using this sorted-summation algorithm since subtraction is performed only once.

We tried both of these methods to see if there was any improvement over a standard implementation using 64-bit floating point. The tests indicated that there was little or no benefit of using arbitrary precision floating-point operations, or the sorted-summation algorithm. Therefore, we used 64-bit floating point operations in our implementation of the analytic method.

Care must be taken when computing the value of $z$ in (3.4). The divisor can be very small and contain few significant digits due to the aforementioned cancellation errors. Such an error will be amplified by the division if the divisor is small. The resulting $z$ value can easily become incorrect and the verification step of the analysis is likely to fail due to an incorrect value for $z$. From equations (3.1) - (3.3), we have a total of three expressions for calculating $z$:

$$z_i = \frac{a_i + b_i x + c_i y + e_i xy}{d_i + f_i x + g_i y + h_i xy} \quad \text{for } i = 1, 2, 3.$$

When a candidate pair $(x, y)$ has been determined, the values are inserted into each of the $z_i$, and the resulting tuple $(x, y, z_i)$ is forwarded to the validation step. The first value to pass the validation step, if any, is marked as a critical point within the cell.

Computing the roots of the resultant polynomial is also a source of numerical error, and will limit the number of critical points discovered. The numerical method used by GSL to find the roots is sensitive to numerical errors, and roots of higher multiplicity are returned as a cluster of simple roots with reduced accuracy.

Another factor that affects the number of critical points the method will locate is the numerical tolerance level used. A critical point will usually not interpolate to exactly zero, but the interpolated value will be smaller than some limit $\epsilon > 0$. The value of $\epsilon$ will depend on the type of vector field being analyzed. For instance, different physical applications will have different requirements as to what value can be considered zero.

### 3.3 Implementation Details

This section covers methods for improving the performance and robustness of the analytic critical point locator. We present a simple method to reduce the computational workload for both the bisection method and the analytic method. For improving the accuracy of the method we introduced a Newton-Raphson iteration step as part of the locating process for a grid cell. Critical points lying on a cell face can result in duplicate critical points after a locating process has been performed. To remove duplicates we perform a duplicate removal process after every grid cell has been analyzed. Finally, there is a discussion of methods for verifying that a critical point is an isolated singularity, and not a point in a connected region of the vector field where every vector is null. A simple example of such a region is a cell face whose vertices are all null-vectors. This will help improve the robustness of both methods.

A flowchart of the algorithm using the bisection method is given in figure 3.3, and a flowchart of the analytic algorithm is provided in figure 3.4. The flowcharts show the individual steps in the analysis algorithm for a single cell. The duplicate-removal process is performed as a post-processing step after all cells in the vector field have been analyzed for critical points.
Figure 3.3: Flowchart of Greene’s bisection algorithm for locating critical points.

Figure 3.4: Flowchart of the analytic algorithm for locating critical points.
3.3.1 Reduction Stage

We have introduced a preprocessing step to see if a cell can contain one or more critical points before the cell is analyzed using the bisection method or the analytic method. This will help increase the performance of both algorithms because of reduced workload. Cells that have been determined not to contain a critical point by this preprocessing step are not be analyzed further.

By the assumption that the field inside the cell is trilinear, the observation is that the range of values for a component inside the cell is limited by the minimum and maximum value of the component at the corner vectors. For example, the $i$-th component must satisfy
\[
\min_{j=0,\ldots,7}(F_i(x_j)) \leq F_i \leq \max_{j=0,\ldots,7}(F_i(x_j)),
\]
where $x_j$ is the coordinate of the $j$-th corner vector with the indexing order for the cell defined as in figure 2.3 on page 8. Because of this, a critical point cannot exist inside a cell where $F_i$ is nonzero and of the same sign at each vertex of the cell.

At each cell we consider the signs of the three components in turn. Should any of the three vector field components have the same sign at every vertex of the cell, the cell is discarded from further analysis since it cannot contain a critical point.

3.3.2 Newton-Raphson Iteration

For the synthetic datasets, we discovered that in most cases, the candidate critical points were close to the actual critical points. The location of those can be determined from the analytic expressions. By this observation, we created the implementation so that it uses the candidate critical point as the initial guess in a multidimensional Newton-Raphson iteration. Since the guess is close to the real critical point, we found that a Newton-Raphson iteration converges quickly to the critical point in most cases. The results for this implementation are discussed in section 3.4. In section 3.2.2 it was noted that the QR-reduction method for finding roots of general polynomials can fail to converge. We employ Newton-Raphson iteration from the center of the cells for these cases to try to locate at least a single critical point.

We also use Newton-Raphson iteration in our implementation of the bisection method. If recursion proceeds to the deepest level, the center of those cells that have a nonzero topological degree are used as initial guess for a Newton-Raphson iteration. The iteration is allowed to pass through the boundary of the cell, possibly locating critical points inside neighboring subcells.

3.3.3 Duplicate Removal

Both the analytic method and the bisection method employ a duplicate-removal process as a post-processing step for the vector field after every grid cell has been analyzed in detail. Let the result set be the set of all critical points after a critical point locating procedure has been performed on a vector field. Consider two neighboring cells with a shared face. If a critical point lies on the shared face, there can be two instances of this critical point in the result set because it can be detected by the search in both of the cells. Also, the numerical root-finder used can converge to the same critical point using different initial locations. We therefore measure the distance between every pairs of critical points to remove duplicates by specifying some minimum distance $\delta$ between them. If two points are separated by a distance less than $\delta$, one of the points is removed from the result set. For the bisection method, a general guideline is to use a value smaller than $1/2^{n+1}$, where $n$ is the maximum number of subdivisions performed.
3.3.4 Validation

A general approach to test if a candidate point is an actual critical point is to insert the point into the trilinear equations to see if it yields null. However, verification of a candidate for a critical point is made more robust by looking at the local region of the critical point. It could be that the critical point is actually part of a connected region where the field is null, and not an isolated critical point. The original algorithm, as presented by Gjøystdal, did not take this into consideration. We have therefore added a validation step to test whether or not a candidate for a critical point is a true isolated critical point.

We considered two different approaches to determine the local region. The first was to see if a close point in some direction, preferably towards the center of the cell to avoid testing a point outside the boundary of the cell, is null. The reliability of this method is limited by the possibility of the critical point being close to the boundary of the null-region. If so, the test for a connected region can fail if we end up just outside the null-region. A more robust approach is to analyze the Jacobian matrix evaluated at the candidate point. For isolated critical points, we must have that the determinant of the Jacobian is non-vanishing. Each candidate critical point is therefore first tested to verify that the length of the vector is zero according to trilinear interpolation, and then the determinant of the Jacobian matrix is calculated to see if it is non-zero. We use this determinant test for validating the critical points in our implementation of both the bisection method and the analytic method.

3.4 Comparison and Results

To compare the two presented critical point locating methods, we use two different types of testing. The first is a simple test were we generate two vector fields were the exact location of the critical points is known. In this part, we also use five datasets derived from numerical simulations. The second type of test is a statistical test where a formula is used to generate vector fields were the critical points are placed at random locations. Both algorithms were implemented in the C++ programming language.

Synthetic Fields and Numerical Simulation Datasets

We start by looking at the synthetic vector fields. The vector fields studied are on the domain \([0, 1] \times [0, 1] \times [0, 1]\), and they are are generated at two different resolutions: \(2 \times 2 \times 2\) and \(50 \times 50 \times 50\) grid points. The lowest resolution discretization is therefore simply a single cell, and locating critical points should be difficult because of the complex topological structure inside the single cell.

From the trilinear equations (3.1) - (3.3) we observe that if the vector field is expressed in the following form

\[
F_1 = (x - x_1)(y - y_1)(z - z_1) \\
F_2 = (x - x_2)(y - y_2)(z - z_2) \\
F_3 = (x - x_3)(y - y_3)(z - z_3),
\]

we can control the number and location of critical points. Critical points will be located at every point \((x_i, y_j, z_k)\) where none of \(i, j\) and \(k\) are equal. By comparing these expressions with the trilinear interpolation equations (3.1) - (3.3), it is evident that the maximum number of critical points that can be located within a single cell is 6.
The first field, $S_2$ (subscript denotes the number of critical points), is given by

\[
F_1 = (x - 0.74)(y - 0.51) \\
F_2 = (x - 0.23)(y - 0.31) \\
F_3 = z - 0.34.
\]

From the expressions it is clear that this vector field has two critical points

\[
c_1 = (0.74, 0.31, 0.34) \quad \text{and} \quad c_2 = (0.23, 0.51, 0.34).
\]

The second field $S_6$ has components

\[
F_1 = (x - 0.11)(y - 0.42)(z - 0.78) \\
F_2 = (x - 0.28)(y - 0.52)(z - 0.82) \\
F_3 = (x - 0.32)(y - 0.67)(z - 0.95),
\]

and has six critical points

\[
c_1 = (0.11, 0.52, 0.95) \\
c_2 = (0.11, 0.67, 0.82) \\
c_3 = (0.28, 0.42, 0.95) \\
c_4 = (0.28, 0.67, 0.78) \\
c_5 = (0.32, 0.42, 0.82) \\
c_6 = (0.32, 0.52, 0.78).
\]

The Isabel\textsuperscript{2} simulation dataset is from a direct numerical simulation of the hurricane Isabel which appeared in the Atlantic Ocean, September 2003. This dataset has a resolution of $500 \times 500 \times 64$. We computed the number of critical points for 4 different time steps. In addition, we analyzed the number of critical points located, and the performance, for a dataset from a simulation of a Kelvin-Helmholtz instability [45]. This dataset has a resolution of $300^3$. The tests were run on a computer with an Intel Core 2 E6600 CPU with 2GB RAM.

The results from the testing are available in table 3.1. For these results, we limited the maximum number of subdivisions for the bisection method to 5. The first important observation is that both methods find and locate the 2 critical points in vector field $S_2$. For $S_6$ we see that Greene’s method identifies a single critical point at the lowest grid resolution while the analytic method finds all 6 critical points. When the resolution is increased, Greene’s method finds all critical points.

For the numerical simulation datasets, the analytic method was able to detect and locate more critical points than Greene’s bisection method for all 5 data sets. On average, the analytic method located 7.5% more critical points compared to the bisection method for the simulation dataset Isabel. For the Kelvin-Helmholtz instability simulation, the analytic method located 18.5% more critical points.

We found that using different compiler optimization levels would affect the number of critical points found. This is due to effects introduced by the compiler when optimization is enabled. An optimization switch allows the compiler to reorder the instructions for a more efficient execution of the program. The results shown in table 3.1 were generated using GCC (GNU Compiler

\textsuperscript{2}Hurricane Isabel data produced by the Weather Research and Forecast (WRF) model, courtesy of NCAR and the U.S. National Science Foundation (NSF).
Locating First-Order Critical Points in Vector Fields

<table>
<thead>
<tr>
<th>Vector Field</th>
<th>Resolution</th>
<th>Greene Number</th>
<th>Time</th>
<th>Analytic Number</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_2$</td>
<td>$2 \times 2 \times 2$</td>
<td>2</td>
<td>$&lt; 0.01$ s</td>
<td>2</td>
<td>$&lt; 0.01$ s</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$50 \times 50 \times 50$</td>
<td>2</td>
<td>0.18 s</td>
<td>2</td>
<td>0.18 s</td>
</tr>
<tr>
<td>$S_6$</td>
<td>$2 \times 2 \times 2$</td>
<td>1</td>
<td>$&lt; 0.01$ s</td>
<td>6</td>
<td>$&lt; 0.01$ s</td>
</tr>
<tr>
<td>$S_6$</td>
<td>$50 \times 50 \times 50$</td>
<td>6</td>
<td>0.18 s</td>
<td>6</td>
<td>0.18 s</td>
</tr>
<tr>
<td>Isabel 1</td>
<td>$500 \times 500 \times 64$</td>
<td>107</td>
<td>24.2 s</td>
<td>113</td>
<td>22.9 s</td>
</tr>
<tr>
<td>Isabel 2</td>
<td>$500 \times 500 \times 64$</td>
<td>230</td>
<td>30.2 s</td>
<td>240</td>
<td>21.9 s</td>
</tr>
<tr>
<td>Isabel 3</td>
<td>$500 \times 500 \times 64$</td>
<td>692</td>
<td>26.4 s</td>
<td>761</td>
<td>26.1 s</td>
</tr>
<tr>
<td>Isabel 4</td>
<td>$500 \times 500 \times 64$</td>
<td>815</td>
<td>30.1 s</td>
<td>901</td>
<td>27.3 s</td>
</tr>
<tr>
<td>Kelvin-Helmholtz</td>
<td>$300 \times 300 \times 300$</td>
<td>27</td>
<td>44.9 s</td>
<td>32</td>
<td>38.1 s</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of the number of critical points located, and the performance of the bisection method compared to the analytic method.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Greene</th>
<th>Analytic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2 \times 2 \times 2$</td>
<td>232</td>
<td>568</td>
</tr>
<tr>
<td>$5 \times 5 \times 5$</td>
<td>485</td>
<td>582</td>
</tr>
<tr>
<td>$10 \times 10 \times 10$</td>
<td>549</td>
<td>596</td>
</tr>
<tr>
<td>$50 \times 50 \times 50$</td>
<td>593</td>
<td>600</td>
</tr>
</tbody>
</table>

Table 3.2: Results for the statistical test on randomized fields. The maximum number of critical points that can be located is 600.

Collection) version 4.1.2 and level 2 optimizations using the flag `-O2`. The number of critical points located for the tests using this level of optimizations were identical to the results using no optimizations, but using level 3 optimization resulted in 0.8% to 5.3% fewer critical points located for each of the time steps of the Isabel dataset. Kelvin-Helmholtz showed similar results.

Performance is better for both the Isabel dataset, and the Kelvin-Helmholtz dataset using the analytic method. However, reducing the number of subdivision levels to 4 results in the bisection method being faster in every case, but with 10% to 18% fewer critical points found.

Statistical Testing

The statistical testing process starts by generating a set of vector fields on the form given by (3.8). A uniform random number generator is used to randomize the location of critical points. To initialize the random number generator, we use the same seed value at every run, so the sequence of analytic expression used in the discretization process is the same for every resolution. Each generated vector field is then analyzed both using Greene’s bisection method and the analytic method. The count of the number of critical points located by each method is computed and can be compared to see which method is able to locate more critical points. This statistical test is performed for fields generated at various resolutions to see how sensitive the methods are to the resolution of the dataset. We use resolutions of 2, 5, 10 and 50 grid points in each direction. 100 randomized vector fields with 6 critical points in each is generated for each resolution. Therefore, the maximum number of critical points that can be detected and located is 600. See table 3.2 for the results. The results show that for the randomized vector fields generated, the analytic method was better at locating critical points independent of resolution. At the highest resolution tested it found all 600 critical points. The bisection method finds less
than half the critical points at the lowest resolution, but at the highest resolution there is only 7 critical points it does not detect. We have made a plot of the number of critical points located as a function of resolution for the two methods, see figure 3.5. This figure shows that the number of critical points located by Greene’s method grows rapidly for resolutions up to 10-15 grid points in each direction, and that there is asymptotic behavior when the resolution is further increased. For the analytic case there are some small oscillations for resolutions lower than 10 grid points, followed by the asymptotic behavior. The comparison of the two methods shows that the analytic approach is better at locating critical points in all tests we have performed, both using synthetic datasets generated from analytic expressions and datasets from a numerical simulation. In [2], results show that different implementations of Greene’s bisection method could add or remove a critical point, and even change the topological degree. There are many ways of choosing the triangulation, and different triangulations can give different results. Together, these types of uncertainties make the bisection method less reliable than the analytic approach presented here.

3.5 Higher-Order Critical Points

The two methods we have looked at have both been concerned with locating first-order critical points. We have defined a critical point to be an isolated point in the vector field where the interpolated vector has a magnitude of zero. The interpolation method used is trilinear interpolation. By limiting the analysis to trilinear interpolation, we artificially reduce the complexity of the topological structure in the field because of the assumption of piecewise linear behavior of the field within the cells. This reduction makes it impossible to detect critical points with a topological degree higher than 1 or lower than -1. To be able to detect and locate higher-order critical points, it is necessary to use a different interpolation scheme that can preserve the topology [35], for instance a polynomial approximation.
Chapter 4

Classification of First-Order Critical Points in Vector Fields

4.1 Motivation

The classification and location of the critical points of a vector field is sufficient to create a simplified model of the field. Such a model may aid a scientist in the understanding of the behavior of the field. Critical points and separating surfaces and lines emanating from such points, divide the field into areas of different types of vector field topology. Studying critical points and their neighborhood can give a clear understanding of the major and most important properties of the field.

4.2 Previous Work

Extensive investigations have been performed regarding the connection between two-dimensional flow fields and systems of linear differential equations [29, 28]. Later, this theory has been generalized and further developed to handle three-dimensional fields [7]. The theory of three-dimensional systems of differential equations on which it builds, has been known for quite some time [32]. The theory of the topology of critical points has then been utilized to visualize vector field topology and compress and reconstruct complex vector fields [17, 31]. Separatrices called saddle connectors were later introduced as means to connect critical points and separate areas of different flow behavior [40]. These connectors were insufficient because they could only emanate from a critical point of the type saddle. An improvement to the technique was therefore conceived in the form of boundary switch curves [44]. The use of topological skeletons was first introduced by Helman and Hesselink in 1989 [16].

Chong et al. have developed a nearly complete scheme for classifying first-order critical points in 3D vector fields [7]. The scheme is based on an analysis of the coefficients $P$, $Q$ and $R$ of the characteristic equation of the Jacobian matrix. In PQR-space, different regions correspond to different combinations of eigenvalues, which again result in different classifications of the critical point. The scheme developed is a table with given classifications for given combinations and values of $P$, $Q$ and $R$. Degenerate cases due to repeated eigenvalues are not completely classified by the scheme, a fact which makes it incomplete. The authors have explained the cases, but due to the limitations in the connection between the PQR-space and the actual eigenset, they are not incorporated in the scheme. The relative values and signs of the eigenvalues can be determined by analyzing $P$, $Q$ and $R$. However, the eigenvectors and geometric multiplicity of the eigenvalues are not revealed. Therefore, using PQR-analysis to resolve the type of critical
points alone does not give a complete classification. All classification schemes of first-order 3D critical points is indirectly based on the eigenvector planes of the Jacobian. The orientation and classification of these individual planes are not revealed by the scheme developed by Chong et al. [7].

4.3 Method

The focus on the visualization of critical points, the current classification schemes, and the confusion of the theory on which they build, forced us to start from the theory of differential equations and design a new scheme. The use of simple synthetic vector fields containing degenerate critical points with repeated eigenvalues, and visualization routines using the orientation of the phase planes, are important factors supporting this decision.

In the validation step of the process of locating critical points, we constrained the valid critical points to be isolated by discarding regions where the vector field vanishes. This constraint eliminates the possibility of eigenvalues being zero. In our classification we have focused on such isolated critical points, and therefore the scheme presented does not cover line-nodes and other classifications which can emerge if lines of critical points are allowed [7, 32].

In the following sections we decompose the process of obtaining a classification of a first-order critical point. As an analytical approach, this process has three major steps which are as follows: Obtain the Jacobian of the critical point, classify the Jacobian as one of seven classes, and finally classify each of the phase planes given by the classification of the Jacobian and pairs of eigenvalues. We have developed a classification scheme which simplifies the analysis of the Jacobian, and implicitly uses the seven classes. These classes are explained in section 4.7.

For our classification scheme, we group the phase planes into three groups. The general phase plane is defined in section 2.4, and the three types are explained in sections 4.6.1, 4.6.2 and 4.6.3. Each type of phase plane has a set of possible phase portraits. So, when the type of a phase plane is established, the set of possible phase portraits and classifications are reduced to those belonging to this type of phase planes.

The classification scheme is presented in detail in the following sections. First, we start by introducing ordinary differential equations, the type of equations on which the analysis is performed.

4.4 Ordinary Differential Equations

Differential equations are equations which contain derivatives. Ordinary differential equations (ODEs) are equations where the derivation is with respect to only one variable, in contrast to partial differential equations (PDEs) where the derivation is performed in more than one variable. Thus, ODEs are functions of only one independent variable, but includes dependent variables which can be regarded as functions of the independent variable. A first-order linear ordinary differential equation can be expressed on the form

\[ x' = p(t)x + g(t), \]

where \( x = x(t) \), \( p(t) \) and \( g(t) \) are functions. This equation is linear because \( x \) only appears in linear combinations, and it is of first order because no higher order derivatives of \( x \) are included.

In this equation the independent variable is \( t \), while \( x \) is a dependent variable and a function of \( t \). We will work with first-order homogeneous equations on the form

\[ x'(t) = ax(t), \quad (4.1) \]
where $a$ is a scalar, and a solution is given by

$$x(t) = Ce^{at}, \quad (4.2)$$

where $C$ is an arbitrary constant.

### 4.4.1 Systems of Coupled Linear ODEs

In section 2.2.5, we showed that in a small region surrounding a critical point the vector field can be approximated by the Jacobian matrix of the critical point. The approximation is given by an equation on the form

$$\mathbf{x}' = A\mathbf{x}, \quad (4.3)$$

which is similar to the first-order linear equation equation (4.1). Since $\mathbf{x}$ is a vector, the solution must also be a vector, so we try the solution

$$\mathbf{x}(t) = \mathbf{v}e^{\lambda t}. \quad (4.4)$$

If we substitute this candidate solution back into (4.3), and divide both sides by the common coefficient $e^{\lambda t}$, we get the equation

$$\lambda \mathbf{v} = A\mathbf{v}, \quad (4.5)$$

which we recognize as the eigenvector equation (2.4) of $A$. This means that when $\lambda$ is an eigenvalue of $A$, and $\mathbf{v}$ is an eigenvector corresponding to $\lambda$, equation (4.4) is a solution to the set of linear ODEs given by equation (4.3).

### 4.4.2 Fundamental Sets

If we have a set of solutions which are all individual solutions to the set of ODEs in (4.3), we can form a linear combination of these by superpositioning

$$\mathbf{x}(t) = \sum_{k=1}^{k} c_k \mathbf{x}_k(t), \quad (4.6)$$

which is also a solution. For equation (4.6) to be a general solution of the set of ODEs, it has to be able to express all possible solutions and initial values. To meet these requirements the set of solutions which the general solution is composed of, has to be a fundamental set of solutions [21]. A fundamental set of solutions is a set of linearly independent solutions, which span the vector space of the governing ODEs. In our case the ODEs have three dependent variables so the vectors have to span $\mathbb{R}^3$ and thus we need a set of linearly independent vectors $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ with dimension $3 \times 1$ [20, 21]. When the solutions form a fundamental set, any point in $\mathbb{R}^3$ can be expressed through a linear combination of the vectors these functions are comprised of.

In this thesis the Jacobian matrix is a $3 \times 3$ matrix and all general solutions discussed are on the form

$$\mathbf{x}(t) = \sum_{k=1}^{3} x_k^*(t) = \sum_{k=1}^{3} f_k(t)\mathbf{v}_k, \quad (4.7)$$

where $\mathbf{v}_k$ is a regular eigenvector, a generalized eigenvector, or the real or imaginary parts of a complex eigenvector, and $f_k(t)$ is a function of $t$. The function $x_k^*(t)$ is not necessarily a solution to the set of ODEs, but is created by combining parts of the different weighted solutions $c_k \mathbf{x}_k(t)$ from equation (4.6). The sum $\mathbf{x}(t)$ is equal in equation (4.6) and equation (4.7). Three constants
$c_0, c_1, c_2$ appear in the functions $f_k(t)$ in such a way that $f_k(0)$ evaluates to $c_k$. These constants are chosen to meet the initial condition given by

$$x(0) = \sum_{k=1}^{3} f_k(0)v_k = \sum_{k=1}^{3} c_kv_k = v_c, \quad (4.8)$$

which states that the solution trajectory of this particular solution will start in the point $v_c$ at $t = 0$. This initial condition is met by choosing the constants $c_k$ so that

$$\begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = v_c, \quad (4.9)$$

As we will see later, forming a fundamental set of solutions will not be trivial when the Jacobian has repeated eigenvalues, that is when $\lambda_i = \lambda_j \neq \lambda_k$ or $\lambda_i = \lambda_j = \lambda_k$, and the eigenvectors are not linearly independent.

### 4.5 3D Phase Portraits

In this section, the Jacobian matrix, and the system of differential equations it represents, are considered to be derived from a critical point in a flow field. The three-dimensional phase portrait of this system represents the flow around the critical point.

The real eigenvectors of the Jacobian are directions of flow. More precisely, they form lines through the critical point defined by the direction of each of the eigenvectors. The sign of the corresponding eigenvalues determine if the the flow along the lines are toward or away from the point. If it is negative, the flow will be in the direction of the critical point, from both sides along the line. If it is positive, it will be directed away from the critical point. With three distinct eigenvalues, there are three such lines of inflow or outflow. The flow between these lines is also defined by the system. If referred to in mathematical context, as solutions of a system of differential equations, these lines are the solution trajectories. The infinite set of solution trajectories are called a phase portrait. The two-dimensional case is explained in section 2.4.

In this thesis, the classification of critical points is deduced using phase planes and two-dimensional phase portraits. The topology of the three-dimensional phase portraits of the critical points will also be explained.

### 4.6 Phase Planes and Solution Trajectories

Choosing the coefficients of the general solution of the set of coupled ODEs so that it consists of a combination of two vectors, yields solution trajectories which will live in the plane spanned by these two vectors. This is because the function evaluated for an arbitrary $t$ will always yield a linear combination of the two vectors, and therefore a point in the plane. These planes will be phase planes. The number of phase planes for a given critical point is determined by the possible combinations of the three solutions defining the general solution, which gives a function comprised of only two vectors. This number varies from one to three for a $3 \times 3$ Jacobian.

The seven classes of the Jacobian, later defined in section 4.7, each have a specific number of phase planes. The type of the individual phase planes are also determined by these classes. This is because each class has a certain form of general solution of the set of ODEs. These forms, and how they are derived, is also presented in section 4.7. By forming linear combinations of the individual solutions the general solution is comprised of, there can be derived a set of functions
involving only two vectors. We have separated these into three different types. The three types of phase planes explained in section 4.6.1, 4.6.2 and 4.6.3 are defined by the type of function the solution trajectories in the plane are given by.

As an example we will derive the two functions defining the two phase planes of a critical point with a class 4 Jacobian. The general solution of the system is

\[ x(t) = \sum_{k=1}^{3} c_k x_k(t) = c_1 e^{\lambda_1 t} v_1 + c_2 e^{\lambda_2 t} v_2 + c_3 e^{\lambda_3 (v_3 + tv_2)}, \quad (4.10) \]

which can be written on the form given in equation (4.7)

\[ x(t) = \sum_{k=1}^{3} f_k(t) v_k = c_1 e^{\lambda_1 t} v_1 + (c_2 + tc_3) e^{\lambda_2 t} v_2 + c_3 e^{\lambda_3 t} v_3, \quad (4.11) \]

where \( v_3 \) is a generalized eigenvector. Because \( c_3 \) appear in both \( f_2 \) and \( f_3 \), \( v_3 \) cannot be combined with \( v_1 \) without also including \( v_2 \) when \( t \neq 0 \). Even though an initial condition given by a point in the plane spanned by \( v_1 \) and \( v_3 \) can be met, the solution trajectory will only pass through the plane. The only phase planes are then the ones spanned by \( (v_1, v_2) \) and \( (v_2, v_3) \). An initial condition given by a point in the plane spanned by one of these pairs will not involve another vector, even when time increases. The functions defining the solution trajectories in each phase plane are

\[ g_1(t) = c_1 e^{\lambda_1 t} v_1 + c_2 e^{\lambda_2 t} v_2 \]
\[ g_2(t) = e^{\lambda_2 ((c_2 + tc_3) v_2 + c_3 v_3)}. \]

The two functions belong to type 1 and type 2 phase planes respectively. In table 4.1, all classes, the corresponding types of phase planes and the solution trajectory functions are listed. The possible classifications of the phase planes for a given class is listed in table 4.2. We proceed to explain the different types of phase planes and their phase portraits.

### 4.6.1 Phase Plane Type 1

These planes have solution trajectories given by the function \( x(t) = c_1 v_1 e^{\lambda_1 t} + c_2 v_2 e^{\lambda_2 t} \). The two eigenvalues can take on any value, and will be equal if the function is constructed by a set of repeated eigenvalues with linearly independent eigenvectors. The possible combinations are \( \lambda_1 < \lambda_2 < 0, \lambda_1 < 0 < \lambda_2, 0 < \lambda_1 < \lambda_2 \) and \( \lambda_1 = \lambda_2 \). The corresponding classifications are attracting node, repelling node, saddle and star node. See figure 4.1 for an illustration of these types of phase planes. The classification of these phase planes are covered in figure 4.5.
4.6.2 Phase Plane Type 2

These planes have solution trajectories given by the function \( x(t) = c_1 v_1 e^{\lambda t} + c_2 e^{\lambda t}(tv_1 + v_2) = e^{\lambda t}((c_1 + c_2 t)v_1 + c_2 v_2) \), and constitute what is referred to as logarithmic nodes. The Jacobian matrices with repeated eigenvalues with algebraic multiplicity higher than the geometric multiplicity are the cases which produce these types of phase planes. An example of a logarithmic node is given in figure 4.2.

\[ x(t) = c_1 v_1 e^{\lambda t} + c_2 e^{\lambda t}(tv_1 + v_2) = e^{\lambda t}((c_1 + c_2 t)v_1 + c_2 v_2) \]

Figure 4.2: Repelling logarithmic node.

4.6.3 Phase Plane Type 3

The only critical points with this type of phase planes are the ones having a Jacobian matrix with complex eigenvalues. This is a Jacobian of class 7, which will be discussed later. The solution trajectories are given by the function

\[
\begin{align*}
x(t) &= c_2 e^{at}(v_a \cos(bt) - v_b \sin(bt)) \\
&+ c_3 e^{at}(v_a \sin(bt) + v_b \cos(bt)) \\
&= e^{at}[c_2(v_a \cos(bt) - v_b \sin(bt)) \\
&+ c_3(v_a \sin(bt) + v_b \cos(bt))],
\end{align*}
\]

where \( a \) is the real part of the eigenvalue, and \( v_a \) and \( v_b \) are the real and imaginary parts of the corresponding eigenvector. This phase plane has two possible classifications: focus or center. Both are comprised of spiraling solution trajectories, but for the center all trajectories are circles or ellipsoids, while a focus has trajectories moving toward or away from the center point as the time \( t \) increases. A focus is said to be compressing if the movement is toward and expanding if it is away from the center. \( a < 0 \) gives a compressing focus, \( a > 0 \) an expanding focus, and \( a = 0 \) a center. The direction of the spiraling movement is given by the sign of \( b \). A positive \( b \) results in a clockwise spiraling trajectory, while a negative \( b \) gives a counter-clockwise motion around an axis defined by the cross product of \( v_a \) and \( v_b \). If \( b = 0 \), the eigenvalue is not complex, and we have a completely different classification. See figure 4.3 for an example of a center and a focus phase plane. The classification these types of phase planes are covered in figure 4.6.
The Jacobian matrix is classified into one of 7 classes according to its eigenvalues.

1. Three real and distinct eigenvalues.
2. One eigenvalue with algebraic multiplicity 3 and geometric multiplicity 3.
3. One eigenvalue with algebraic multiplicity 1, and one eigenvalue with algebraic multiplicity 2 and geometric multiplicity 2.
4. One eigenvalue with algebraic multiplicity 3 and geometric multiplicity 2.
5. One eigenvalue with algebraic multiplicity 3 and geometric multiplicity 1.
6. One eigenvalue with algebraic multiplicity 1, and one eigenvalue with algebraic multiplicity 2 and geometric multiplicity 1.
7. Complex eigenvalues.

Each class has its distinct Jordan normal form, and a Jacobian can be classified by determining which of the seven forms it produces. This form can be used as a mean of separating the different types of Jacobian matrices [7, 32]. In this thesis, the focus is on the multiplicity of the eigenvalues, and the Jordan matrix is only used as a mean of explaining the classes already defined. An introduction to the Jordan normal form is given in section 2.5, and the seven Jordan normal forms and the corresponding classes are given in table 4.3. The validity of the different classes and types of phase planes, in regards to the pressens of repeated eigenvalues and a defective Jacobian matrix, is visualized in figure 4.12.

We now proceed to explain how the orientation of the phase planes, and the phase portraits they contain, are calculated for each of the seven classes. The classes are grouped in three, according to the presence of repeated or complex eigenvalues.

### 4.7.1 Distinct Eigenvalues

Class 1 is the only class with three distinct eigenvalues. When the eigenvalues are distinct, the eigenvectors will form a linearly independent set [24]. Thus, the pairs of eigenvalues and eigenvectors can be used directly to form a fundamental set of solutions \( \{ v_1 e^{\lambda_1 t}, v_2 e^{\lambda_2 t}, v_3 e^{\lambda_3 t} \} \). From equation (4.9) it is evident that if an initial point is in a plane spanned by two of the three vectors, the coefficient of the third vector will be zero, and the solution trajectories will remain in the plane. The combinations of the three vectors result in three planes of type 1, defined by sets of two vector functions on the form \( v e^{\lambda t} \).
4.7.2 Repeated Eigenvalues

As described in section 2.3.1, an eigenvalue of multiplicity $k$ can have as many as $k$ and as few as one linearly independent eigenvectors corresponding to it. This number depends on the form of the corresponding matrix. For example, computing the eigenvectors of the matrices $A$ and $B$ using GNU Octave results in

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow A_v = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$  

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow B_v = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

where the eigenvectors of $A$ and $B$ are the columns of $A_v$ and $B_v$. We cannot use the columns of $B_v$ directly to form a fundamental set of solutions to the equation $x' = Bx$. What we need is the vectors of (4.6) to form a complete basis for $\mathbb{R}^3$. As explained in section 2.3.2, a complete basis of a defective matrix can be formed using generalized eigenvectors.

Classes 2 and 3 have repeated eigenvalues, but the total geometric multiplicity is 3 and the eigenvectors of the matrix form a linearly independent set of size 3. The solutions of these two classes are given by functions on the same form as those of class 1. What differs is that for some of the planes, the eigenvalues will be equal. Class 2 produces three planes with solution trajectories given by the function $x(t) = c_1v_1e^{\lambda t} + c_2v_2e^{\lambda t}$. These are referred to as star nodes and will be explained later. Class 3 has three planes of type 1 as well, but only one of these has equal eigenvalues and solution trajectories which form a star node. The structure of the trajectories in the other two planes can be established by checking the classifications of a phase plane of type 1 for the respective eigenvalues.

Classes 4, 5 and 6 all have repeated eigenvalues. What separates these classes from 2 and 3 is a total geometric multiplicity less than 3, which necessitates the use of generalized eigenvectors. To make use of the generalized eigenvectors, a solution on the form

$$x(t) = v_1te^{\lambda t} + v_2e^{\lambda t}, \quad (4.12)$$

is chosen, and substituted back into (4.3):

$$x'(t) = Ax(t)$$

$$v_1 e^{\lambda t} + \lambda v_1 e^{\lambda t} + \lambda v_2 e^{\lambda t} = A(v_1 te^{\lambda t} + v_2 e^{\lambda t})$$

$$(v_1 + \lambda v_2)e^{\lambda t} + \lambda v_1 te^{\lambda t} = Av_1 te^{\lambda t} + Av_2 e^{\lambda t}.$$ 

For this to be true, the coefficients of $e^{\lambda t}$ on the left side of the equation must be equal to the coefficients of $e^{\lambda t}$ on the right side of the equation. This requirement must also be fulfilled for $te^{\lambda t}$. For $e^{\lambda t}$ the equation is

$$v_1 + \lambda v_2 = Av_2$$

$$v_1 = (A - \lambda I)v_2, \quad (4.13)$$

and the equation for the terms involving $te^{\lambda t}$ is

$$\lambda v_1 = Av_1. \quad (4.14)$$
The vector $v_1$ has to be an eigenvector of $A$, while we recognize (4.13) as (2.7), making $v_2$ a generalized eigenvector.

Classes 6 and 4 both have a total geometric multiplicity of 2, and the use of a single generalized eigenvector is required to create a fundamental set of solutions. In both these cases the general solution will be

$$\mathbf{x}(t) = c_1v_1e^{\lambda_1 t} + c_2v_2e^{\lambda_2 t} + c_3e^{\lambda_3 t}(tv_2 + v_3),$$

(4.15)

but for class 4 all eigenvalues will be equal, while for class 4 only $\lambda_2$ and $\lambda_3$ will be. Both classes have two planes containing solution trajectories; one of type 1 and one of type 2.

In the case of Class 5, two generalized eigenvectors are needed to form a general solution of the set of ODEs. The solutions of equation (4.3) for $\lambda$ with algebraic multiplicity $m$ and geometric multiplicity 1 is given by Kohler and Johnson [21]:

$$\mathbf{x}_k = e^{\lambda t}(v_k + t v_{k-1} + \cdots + \frac{t^{k-1}}{(k-1)!}v_1) = e^{\lambda t} \sum_{j=1}^{k} \frac{t^{k-j}}{(k-j)!}v_j.$$  

(4.16)

Using this formula for a class 5 system of ODEs results in the three solutions

$$\begin{align*}
\mathbf{x}_1 &= e^{\lambda t}v_1 \\
\mathbf{x}_2 &= e^{\lambda t}(v_2 + tv_1) \\
\mathbf{x}_3 &= e^{\lambda t}(v_3 + tv_2 + \frac{t^2}{2}v_1).
\end{align*}$$

(4.17)

The solution trajectories given by $\mathbf{x}_3$ will not stay in a plane since three vectors are involved. The plane spanned by $v_1$ and $v_2$, which is of type 2, is the only plane which will contain solution trajectories, and the trajectories are given by the function $\mathbf{x}(t) = e^{\lambda t}(c_2v_2 + (c_2t + c_1)v_1)$. This phase portrait represents a logarithmic node.

Indeterminate Eigenvectors

When a matrix has repeated eigenvalues, but still has a complete set of linearly independent eigenvectors, all three eigenvectors are not uniquely determined. As explained in section 2.3.1, the geometric multiplicity $m_g$ of an eigenvalue equals the dimension of the eigenspace of this eigenvalue. The eigenspace is a subspace of $\mathbb{R}^n$ for an $n \times n$ matrix. For an eigenvalue of a $3 \times 3$ matrix with $m_g = 2$, the eigenspace is a two-dimensional subspace of $\mathbb{R}^3$. Then, any vector in this subspace is a valid eigenvector. Any basis of an $n$-dimensional subspace is always a set of linearly independent vectors, so a set of $m_g$ linearly independent eigenvectors is always obtainable.

When the eigenvectors are not uniquely determined, neither are the phase planes. In the case of class 1 and 2 Jacobians, which are the only Jacobian classes where the eigenvalues have $m_g > 1$, all phase planes are not uniquely determined. What is not uniquely determined is not the type of phase portraits, maintaining the uniqueness of the classification, but the orientation of the phase planes. For class 2, which has one eigenvalue with $m_g = 3$, any vector in $\mathbb{R}^3$ is an eigenvector. That means that any set of three linearly independent vectors in $\mathbb{R}^3$ will constitute a fundamental set of solutions, and any plane through the critical point will contain a star node. Class 3 has one eigenvalue with $m_g = 2$. The phase plane spanned by the two eigenvectors corresponding to this eigenvalue will be fixed, and has the same orientation as the eigenspace of this eigenvalue. This is because any two eigenvectors spanning it will come from the same two-dimensional subspace of $\mathbb{R}^3$. In our classification scheme, there are three phase planes in the
classification of critical points with a Jacobian belonging to class 3. In fact, any plane spanned by the uniquely determined eigenvector, and an arbitrary eigenvector from the eigenspace of the repeated eigenvalue, will be a phase plane. To limit the number of planes, the three eigenvectors closest to forming an orthogonal basis are chosen.

4.7.3 Complex Eigenvalues

A solution to the set of equations \( \mathbf{x}' = A \mathbf{x} \) is \( \mathbf{x}(t) = \mathbf{v} e^{\lambda t} \), where \( \mathbf{v} \) is an eigenvector of \( A \) corresponding to the eigenvalue \( \lambda \). If \( \lambda \) and \( \mathbf{v} \) are complex, we can produce a set of real solutions \( \mathbf{x}(t) \) by separating the system into a real and an imaginary part \( \mathbf{x}(t) = \mathbf{u}(t) + i \mathbf{v}(t) \), where \( \mathbf{u}(t) \) and \( \mathbf{v}(t) \) are real solutions to \( \mathbf{x}' = A \mathbf{x} \) [21]. For a \( 3 \times 3 \) matrix \( A \) with complex eigenvalues, a fundamental set of three solutions can be derived. Let \( \lambda_1 = c \) be the single real eigenvalue, and \( \lambda_2 = a + bi \) and \( \lambda_3 = a - bi \) the complex conjugate pair of eigenvalues for matrix \( A \). The corresponding eigenvectors are split into real and imaginary parts:

\[
\mathbf{v}_2 = \mathbf{v}_a + i \mathbf{v}_b
\]

\[
\mathbf{v}_3 = \overline{\mathbf{v}}_2.
\]

Now, let

\[
\mathbf{x}_1(t) = \mathbf{v}_1 e^{\lambda_1 t}
\]

\[
\mathbf{x}_2(t) = \mathbf{v}_2 e^{\lambda_2 t} = (\mathbf{v}_a + i \mathbf{v}_b) e^{\lambda_2 t}
\]

\[
\mathbf{x}_3(t) = \mathbf{v}_3 e^{\lambda_3 t} = (\mathbf{v}_a - i \mathbf{v}_b) e^{\lambda_3 t}
\]

be the solutions to the system of equations, where the two last equations are complex. Using Euler’s formula, solution \( \mathbf{x}_2(t) \) can be rewritten as

\[
\mathbf{x}_2(t) = \mathbf{v}_2 e^{\lambda_2 t}
\]

\[
= (\mathbf{v}_a + i \mathbf{v}_b) e^{(a+bi)t}
\]

\[
= (\mathbf{v}_a + i \mathbf{v}_b) e^{at} e^{i bt}
\]

\[
= (\mathbf{v}_a + i \mathbf{v}_b) e^{at} (\cos(b t) + i \sin(b t))
\]

\[
= e^{at} (\mathbf{v}_a \cos(b t) + i \mathbf{v}_b \sin(b t)) + i e^{at} (\mathbf{v}_a \sin(b t) + \mathbf{v}_b \cos(b t))
\]

\[
= e^{at} (\mathbf{v}_a \cos(b t) - \mathbf{v}_b \sin(b t)) + i e^{at} (\mathbf{v}_a \sin(b t) + \mathbf{v}_b \cos(b t))
\]

\[
= \mathbf{u}(t) + i \mathbf{v}(t),
\]

where we have split the solution into a real and an imaginary part:

\[
\mathbf{u}(t) = e^{at} (\mathbf{v}_a \cos(b t) - \mathbf{v}_b \sin(b t))
\]

\[
\mathbf{v}(t) = e^{at} (\mathbf{v}_a \sin(b t) + \mathbf{v}_b \cos(b t)).
\]

We now have a fundamental set of solutions from which we can construct the general solution. The general solution for the system is given by

\[
\mathbf{x}(t) = c_1 \mathbf{x}_1(t) + c_2 \mathbf{u}(t) + c_3 \mathbf{v}(t)
\]

\[
= c_1 \mathbf{v}_1 e^{\lambda_1 t}
\]

\[
+ c_2 e^{at} (\mathbf{v}_a \cos(b t) - \mathbf{v}_b \sin(b t))
\]

\[
+ c_3 e^{at} (\mathbf{v}_a \sin(b t) + \mathbf{v}_b \cos(b t)).
\]

From equation (4.18) we see that the only way to get a plane spanned by pairs of the three vectors \( \mathbf{v}_1, \mathbf{v}_a \) and \( \mathbf{v}_b \) is to choose the coefficients so that \( c_1 = 0 \), and at least one of \( c_2 \) or \( c_3 \) is non-zero, since \( c_2 \) and \( c_3 \) appear in products containing both \( \mathbf{v}_a \) and \( \mathbf{v}_b \).
4.8 General Classification Scheme

The critical point classification scheme as devised by Chong et al. [7] uses a PQR-chart from which it is difficult to get an intuitive understanding of the relationship between the coefficients P, Q and R, and the corresponding classification. Each class of critical points is given explicitly by the values of P, Q and R, but does not give a way to group similar types of critical points. The orientation and classification of the separate phase planes is not given.

We have constructed a classification scheme based on the analysis of individual phase planes of a critical point. The method is also able to detect and handle degenerate cases. Degenerate cases occur when the geometric multiplicity of an eigenvalue is less than the corresponding algebraic multiplicity. As a result of the analysis of the individual planes, the orientation of the phase planes is established. The seeding template presented in section 5.1 is dependent on the orientation and classification of the phase planes. Tangent LIC, as presented in section 5.3 requires the orientation of the phase planes to depict the correct phase portraits. Therefore, the classification scheme presented in this thesis is more suited than a PQR-scheme when a detailed visualization of the critical points are to be performed.

The classification is performed as follows. If the Jacobian has complex eigenvalues, there will be one phase plane of type 3. This can be classified by analyzing the relations of the eigenvalues. Section 4.6.3 and table 4.6 explains this process. If the eigenvalues are real, then each eigenvalue and its corresponding eigenvector(s) is checked to see if the geometric multiplicity is equal to the algebraic multiplicity. If it is, each eigenvector in the set corresponding to the eigenvalue is paired with the eigenvalue and set aside to later form a phase plane of type 1. There has to be at least two such eigenpairs to form a type 1 phase plane. A class 5 Jacobian has only got one eigenvector, and the eigenpair set aside is never used. This is the only case where this happens. If the geometric multiplicity of an eigenvalue is less than the algebraic multiplicity, all linearly independent eigenvectors corresponding to this eigenvalue are paired with the eigenvalue and set aside to form phase planes of type 1. In addition, one eigenvector and the corresponding generalized eigenvector are used in combination with the eigenvalue to form a phase plane of type 2.

Flowcharts illustrating the classification of the phase planes are given in figures 4.4, 4.5 and 4.6. The first of these figures describes the analysis of the Jacobian of a critical point given the set of eigenvalues and corresponding eigenvectors. Depending on the properties of this eigenset, each of the extracted phase planes are passed on to the classification step for phase planes given in figures 4.5 and 4.6. Below we give an explanation to each of the numerical annotations in figure 4.4.

1

When the eigenset has complex eigenvalues, the phase planes are limited to be of type 3.

2

If the geometric multiplicity of the eigenvalue is lower than the algebraic multiplicity, a tuple comprised of the eigenvalue, an eigenvector and the corresponding generalized eigenvector is added to the set of generalized solutions.

3

Each vector in the set of linearly independent eigenvectors of the current eigenvalue is paired with the eigenvalue to form a tuple, and registered in a set holding this type of combinations.
Eigenpairs with identical geometric and algebraic multiplicity are simply passed to the set of regular eigenpairs.

Each tuple of generalized solutions is passed on to type 2 phase plane classification.

All possible pairs of eigenpairs are passed on to type 1 phase plane classification.

### 4.9 The 3D Topology

In this chapter we have concentrated on the phase planes of the critical points. The topology of a critical point in three-dimensional vector fields is in every aspect three-dimensional, but the simplification introduced by analyzing the phase planes, is by all means mathematically correct. However, it is not unusual to use the term repelling saddle for a saddle/saddle/node combination, where the node is repelling [40]. For complex eigenvalues, we have used some of these labels already. In this section we explain some of the 3D classifications, and use images from visualizations to help improve the understanding. The visualizations are produced using our hybrid approach as presented in section 5.4.

#### 4.9.1 Real and Distinct Eigenvalues

When the eigenvalues are real and distinct, we have three linearly independent eigenvectors defining the direction of direct inflow and outflow as explained in section 4.5. When restricting
Real eigenvalues: $\lambda_1, \lambda_2$

Figure 4.5: Type 1 phase plane classification flowchart.

Real eigenvalue: $\lambda_1$  Complex eigenvalues: $\lambda_2 = a \pm bi$

Figure 4.6: Type 3 phase plane classification flowchart.
the eigenvalues to be non-zero, as we have done in this thesis by only considering isolated critical points, the number of inflow and outflow directions can be any of four combinations. If \( f_i \) is defined to be the number of inflow directions, the number of outflow directions is \( f_o = 3 - f_i \). The four possibilities are explained in the following subsections.

**Attracting Node**

If all eigenvalues are negative, there is inflow in all directions. This is shown in figure 4.7.

**Repelling Node**

If all eigenvalues are positive, there is outflow in all directions. This is shown in figure 4.7.

**Attracting Saddle**

If two eigenvalues are negative, and the last is positive, the point is an attracting saddle. This type is defined by inflow in one plane, and a outflow along a single line. This causes the flow to asymptote to the plane containing the attracting node. The plane-wise classification is saddle/saddle/attracting node. A point having this topology is visualized in figure 4.8.

**Repelling Saddle**

A repelling saddle is the reverse of an attracting saddle. There is one plane of outflow, and one line along which direct inflow occurs. The plane-wise classification is saddle/saddle/repelling node. A point of this type is visualized in figure 4.8.

### 4.9.2 Complex Eigenvalues

**Attracting Focus**

When the real part of the complex eigenvalues is negative, the spiral in the phase plane, spanned by the real and the imaginary parts of the complex eigenvector, moves toward the center. This makes the focus attracting. What separates an attracting focus from an attracting focus saddle is a negative real eigenvalue. An attracting focus has inflow in the phase plane spanned by the real and the imaginary parts of the complex eigenvector, and an inflow from both directions of the real eigenvector. A point of this type is visualized in figure 4.9.

**Repelling Focus**

A repelling focus is identical to an attracting focus, except all flow directions are reversed. A point of this type is visualized in figure 4.9.

**Attracting Saddle Focus**

An attracting saddle focus is an attracting focus where the real eigenvalue is positive. There is inflow in the phase plane, and an outflow in both directions of the real eigenvector. A point of this type is visualized in figure 4.10.

**Repelling Saddle Focus**

A repelling saddle focus is an attracting saddle focus with all flows reversed. See figure 4.10 for an example.
Figure 4.7: Left: Attracting node. Right: Repelling node. The flow directions are color coded. Blue is inflow and red is outflow.

**Attracting Center**

When the real part of the complex eigenvalues is zero, the solution trajectories in the phase plane are closed curves. A center has no in- or outflow in the phase plane, but there is in- or outflow along the direction of the real eigenvector. An attracting center has inflow along this line. A point of this type is visualized in figure 4.11.

**Repelling Center**

A repelling center has outflow along the line defined by the real eigenvector. A point of this type is visualized in figure 4.11.

**4.10 Conclusion**

The classification of first-order critical points in 3D vector fields is dependent on properties of the Jacobian matrix evaluated at the critical point. In this thesis, we have presented a classification scheme in which we analyze the topology of critical points based on the eigenvalues of the Jacobian, and their algebraic and geometric multiplicities. The scheme is complete for isolated critical points. By complete we mean that the scheme also covers degenerate cases. When there exists a system for determining the eigenvalues, and the multiplicity of these, the classification of the critical point can easily be established using the scheme presented in this thesis.
Figure 4.8: Left: Attracting saddle. Right: Repelling saddle. The flow directions are color coded. Blue is inflow and red is outflow.

Figure 4.9: Left: Attracting focus. Right: Repelling focus. The flow directions are color coded. Blue is inflow and red is outflow.
Figure 4.10: Left: Attracting saddle focus. Right: Repelling saddle focus. The flow directions are color coded. Blue is inflow and red is outflow.

Figure 4.11: Left: Attracting center. Right: Repelling center. The flow directions are color coded. Blue is inflow and red is outflow. The artifacts along the diagonal are due to a line of zero magnitude in the flow tangential to the plane.
Figure 4.12: A diagram of the validity of the different Jacobian classes and phase plane types. The purple and the two green boxes are the types of phase planes. From the diagram it is apparent that a type 3 phase plane can only exist where the eigenvalues are distinct (and complex), while a type 1 phase plane can exist when the eigenvalues are distinct, when the eigenvalues are repeated and the Jacobian is defective, and when the eigenvalues are repeated but the Jacobian has a full set of linearly independent eigenvectors (not defective). The numbers marked with x are the number of phase planes of the underlying type the current class has. A class 1 Jacobian has three phase planes of type 1, while a class 4 Jacobian has one of type 1 and one of type 2. A class 5 Jacobian can exist if the eigenvalues are repeated and the Jacobian is defective, a class 3 Jacobian can exist if the eigenvalues are repeated and the Jacobian is not defective.
### Table 4.1: Summary of the classification scheme.

The column C means complex eigenvalues, D is Defective Jacobian, and T is the type of phase planes. In the vector sets, \( \mathbf{v_g} \) is a generalized eigenvector. The Solutions column gives the functions defining the solution trajectories in the respective planes. The number of rows used for each class is the number of existing phase planes, except for in the case of class 7, where there is only one.

<table>
<thead>
<tr>
<th>Class</th>
<th>C</th>
<th>D</th>
<th>( m_a )</th>
<th>( m_g )</th>
<th>Eigenvalues</th>
<th>Vectors</th>
<th>T</th>
<th>Solutions</th>
</tr>
</thead>
</table>
| 1     | no| no| 1,1,1  | 1,1,1  | \( \lambda_i \neq \lambda_j \) for \( i \neq j \) | \( \{ \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \} \) | 1 | \( c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_2 \mathbf{v}_2 e^{\lambda_2 t} \)  
|       |   |   |         |        |             |         |   | \( c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_3 \mathbf{v}_3 e^{\lambda_3 t} \)  
|       |   |   |         |        |             |         |   | \( c_2 \mathbf{v}_2 e^{\lambda_2 t} + c_3 \mathbf{v}_3 e^{\lambda_3 t} \) |
| 2     | no| no| 3       | 3      | \( \lambda_1 = \lambda_2 = \lambda_3 = \lambda_r \) | \( \{ \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \} \) | 1 | \( c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_2 \mathbf{v}_2 e^{\lambda_2 t} \)  
|       |   |   |         |        |             |         |   | \( c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_3 \mathbf{v}_3 e^{\lambda_3 t} \)  
|       |   |   |         |        |             |         |   | \( c_2 \mathbf{v}_2 e^{\lambda_2 t} + c_3 \mathbf{v}_3 e^{\lambda_3 t} \) |
| 3     | no| no| 1,2     | 1,2    | \( \lambda_1 \neq \lambda_2 = \lambda_3 = \lambda_r \) | \( \{ \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \} \) | 1 | \( c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_2 \mathbf{v}_2 e^{\lambda_2 t} \)  
|       |   |   |         |        |             |         |   | \( c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_3 \mathbf{v}_3 e^{\lambda_3 t} \)  
|       |   |   |         |        |             |         |   | \( c_2 \mathbf{v}_2 e^{\lambda_2 t} + c_3 \mathbf{v}_3 e^{\lambda_3 t} \) |
| 4     | no| yes| 3       | 2      | \( \lambda_1 = \lambda_2 = \lambda_3 = \lambda_r \) | \( \{ \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_g \} \) | 1 | \( c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_2 \mathbf{v}_2 e^{\lambda_2 t} \)  
|       |   |   |         |        |             |         |   | \( c_2 \mathbf{v}_2 e^{\lambda_2 t} + c_3 \mathbf{v}_3 e^{\lambda_3 t} (t \mathbf{v}_2 + \mathbf{v}_g) \) |
| 5     | no| yes| 3       | 1      | \( \lambda_1 = \lambda_2 = \lambda_3 = \lambda_r \) | \( \{ \mathbf{v}, \mathbf{v}_g_1, \mathbf{v}_g_2 \} \) | 2 | \( c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_2 \mathbf{v}_2 e^{\lambda_2 t} (t \mathbf{v}_1 + \mathbf{v}_g) \)  
| 6     | no| yes| 1,2     | 1,1    | \( \lambda_1 \neq \lambda_2 = \lambda_3 = \lambda_r \) | \( \{ \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_g \} \) | 1 | \( c_1 \mathbf{v}_1 e^{\lambda_1 t} + c_2 \mathbf{v}_2 e^{\lambda_2 t} \)  
|       |   |   |         |        |             |         |   | \( c_2 \mathbf{v}_2 e^{\lambda_2 t} + c_3 \mathbf{v}_3 e^{\lambda_3 t} (t \mathbf{v}_2 + \mathbf{v}_g) \) |
| 7     | yes| no| 1,1,1   | 1,1,1  | \( \lambda_1 = \lambda, \lambda_2 = a + bi, \)  
|       |   |   |         |        | \( \lambda_3 = a - bi \) | \( \{ \mathbf{v}_1, \mathbf{v}_a + i \mathbf{v}_b, \)  
|       |   |   |         |        | \( \mathbf{v}_a - i \mathbf{v}_b \} \) | 3 | \( c_2 e^{at} (\mathbf{v}_a \cos (bt) - \mathbf{v}_b \sin (bt)) \)  
<p>|       |   |   |         |        | ( + c_3 e^{at} (\mathbf{v}_a \sin (bt) + \mathbf{v}_b \cos (bt)) ) |</p>
<table>
<thead>
<tr>
<th>Class</th>
<th>Eigenvalues</th>
<th>Plane 1 ($\lambda_1, \lambda_2$)</th>
<th>Plane 2 ($\lambda_1, \lambda_3$)</th>
<th>Plane 3 ($\lambda_2, \lambda_3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\lambda_1 &lt; \lambda_2 &lt; \lambda_3 &lt; 0$</td>
<td>att. node</td>
<td>att. node</td>
<td>att. node</td>
</tr>
<tr>
<td></td>
<td>$\lambda_1 &lt; \lambda_2 &lt; 0 &lt; \lambda_3$</td>
<td>att. node</td>
<td>saddle</td>
<td>saddle</td>
</tr>
<tr>
<td></td>
<td>$\lambda_1 &lt; 0 &lt; \lambda_2 &lt; \lambda_3$</td>
<td>saddle</td>
<td>saddle</td>
<td>rep. node</td>
</tr>
<tr>
<td></td>
<td>$0 &lt; \lambda_1 &lt; \lambda_2 &lt; \lambda_3$</td>
<td>rep. node</td>
<td>rep. node</td>
<td>rep. node</td>
</tr>
<tr>
<td>2</td>
<td>$\lambda_1 = \lambda_2 = \lambda_3 &lt; 0$</td>
<td>att. star node</td>
<td>att. star node</td>
<td>att. star node</td>
</tr>
<tr>
<td></td>
<td>$0 &lt; \lambda_1 = \lambda_2 = \lambda_3$</td>
<td>rep. star node</td>
<td>rep. star node</td>
<td>rep. star node</td>
</tr>
<tr>
<td>3</td>
<td>$\lambda_1, \lambda_r &lt; 0$</td>
<td>att. node</td>
<td>att. node</td>
<td>att. star node</td>
</tr>
<tr>
<td></td>
<td>$\lambda_1, \lambda_r &gt; 0$</td>
<td>rep. node</td>
<td>rep. node</td>
<td>rep. star node</td>
</tr>
<tr>
<td></td>
<td>$\lambda_1 &lt; 0 &lt; \lambda_r$</td>
<td>saddle</td>
<td>saddle</td>
<td>rep. star node</td>
</tr>
<tr>
<td></td>
<td>$\lambda_r &lt; 0 &lt; \lambda_1$</td>
<td>saddle</td>
<td>saddle</td>
<td>rep. star node</td>
</tr>
<tr>
<td>4</td>
<td>$\lambda_1 = \lambda_2 = \lambda_3 &lt; 0$</td>
<td>att. star node</td>
<td>att. log. node</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$0 &lt; \lambda_1 = \lambda_2 = \lambda_3$</td>
<td>rep. star node</td>
<td>rep. log. node</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>$\lambda_1 = \lambda_2 = \lambda_3 &lt; 0$</td>
<td>att. log. node</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$0 &lt; \lambda_1 = \lambda_2 = \lambda_3$</td>
<td>rep. log. node</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>$\lambda_1, \lambda_r &lt; 0$</td>
<td>att. node</td>
<td>att. log. node</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$0 &lt; \lambda_1, \lambda_r$</td>
<td>rep. node</td>
<td>rep. log. node</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$\lambda_1 &lt; 0 &lt; \lambda_r$</td>
<td>saddle</td>
<td>rep. log. node</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$\lambda_r &lt; 0 &lt; \lambda_1$</td>
<td>saddle</td>
<td>att. log. node</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>$\lambda_1, a &lt; 0$</td>
<td>att. focus</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$0 &lt; \lambda_1, a$</td>
<td>rep. focus</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$\lambda_1 &lt; 0 &lt; a$</td>
<td>rep. saddle focus</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$a &lt; 0 &lt; \lambda_1$</td>
<td>att. saddle focus</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$a = 0, \lambda_1 &lt; 0$</td>
<td>att. center</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>$a = 0, 0 &lt; \lambda_1$</td>
<td>rep. center</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.2: Overview of possible classifications. For complex eigenvalues we have defined $\lambda_1$ to be the real eigenvalue, and $\lambda_2 = a + bi$ and $\lambda_3 = a - bi$ to be the complex conjugate pair of eigenvalues. For repeated eigenvalues, the value is denoted by $\lambda_r$. Attracting is abbreviated by att., repelling by rep., and logarithmic by log.
<table>
<thead>
<tr>
<th>Class</th>
<th>Jordan Normal Form</th>
<th>Eigenvectors</th>
<th>General Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\begin{pmatrix} \lambda_1 &amp; 0 &amp; 0 \ 0 &amp; \lambda_2 &amp; 0 \ 0 &amp; 0 &amp; \lambda_3 \end{pmatrix}$</td>
<td>${v_1^1, v_1^2, v_1^3}$</td>
<td>$x(t) = c_1v_1^1e^{\lambda_1 t} + c_2v_1^2e^{\lambda_2 t} + c_3v_1^3e^{\lambda_3 t}$</td>
</tr>
<tr>
<td>2</td>
<td>$\begin{pmatrix} \lambda_1 &amp; 0 &amp; 0 \ 0 &amp; \lambda_1 &amp; 0 \ 0 &amp; 0 &amp; \lambda_1 \end{pmatrix}$</td>
<td>${v_1^1, v_2^1, v_3^1}$</td>
<td>$x(t) = e^{\lambda_1 t}(c_1v_1^1 + c_2v_2^1 + c_3v_3^1)$</td>
</tr>
<tr>
<td>3</td>
<td>$\begin{pmatrix} \lambda_1 &amp; 0 &amp; 0 \ 0 &amp; \lambda_2 &amp; 0 \ 0 &amp; 0 &amp; \lambda_2 \end{pmatrix}$</td>
<td>${v_1^1, v_1^2, v_2^2}$</td>
<td>$x(t) = e^{\lambda_1 t}c_1v_1^1 + e^{\lambda_2 t}(c_2v_1^2 + c_3v_2^2)$</td>
</tr>
<tr>
<td>4</td>
<td>$\begin{pmatrix} \lambda_1 &amp; 0 &amp; 0 \ 0 &amp; \lambda_1 &amp; 1 \ 0 &amp; 0 &amp; \lambda_1 \end{pmatrix}$</td>
<td>${v_1^1, v_2^1, v_{g1}}$</td>
<td>$x(t) = e^{\lambda_1 t}(c_1v_1^1 + (c_2 + c_3 t)v_2^1 + c_3v_{g1})$</td>
</tr>
<tr>
<td>5</td>
<td>$\begin{pmatrix} \lambda_1 &amp; 1 &amp; 0 \ 0 &amp; \lambda_1 &amp; 1 \ 0 &amp; 0 &amp; \lambda_1 \end{pmatrix}$</td>
<td>${v_1^1, v_{g1}, v_{g2}}$</td>
<td>$x(t) = e^{\lambda_1 t}((c_1 + c_2 t + c_3 t^2)v_1^1 + (c_2 + c_3 t)v_{g1} + c_3v_{g2})$</td>
</tr>
<tr>
<td>6</td>
<td>$\begin{pmatrix} \lambda_1 &amp; 0 &amp; 0 \ 0 &amp; \lambda_2 &amp; 1 \ 0 &amp; 0 &amp; \lambda_2 \end{pmatrix}$</td>
<td>${v_1^1, v_1^2, v_{g1}}$</td>
<td>$x(t) = c_1v_1^1e^{\lambda_1 t} + e^{\lambda_2 t}((c_2 + c_3 t)v_1^2 + c_3v_{g1})$</td>
</tr>
<tr>
<td>7</td>
<td>$\begin{pmatrix} a &amp; -b &amp; 0 \ b &amp; a &amp; 0 \ 0 &amp; 0 &amp; \lambda_1 \end{pmatrix}$</td>
<td>${v_1, v_c, v_c^\perp}$</td>
<td>$x(t) = c_1v_1e^{\lambda_1 t} + c_2e^{at}(v_a \cos(bt) - v_b \sin(bt)) + c_3e^{at}(v_a \sin(bt) + v_b \cos(bt))$</td>
</tr>
</tbody>
</table>

Table 4.3: The various classes of Jacobian matrices, their Jordan forms, set of eigenvectors and general solutions. For repeated eigenvalues we have used the notation $v_i^j$ to denote the $i$-th eigenvector belonging to eigenvalue $\lambda_j$. Vectors on the form $v_{g1}$ are generalized eigenvectors. For class 7, the single real eigenvector is $v_1$, and the complex conjugate pair of eigenvectors is $v_c = v_a + iv_b$ and $v_c^\perp = v_a - iv_b$, with corresponding eigenvalues $\lambda = a \pm bi$. 
Chapter 5

Visualization of First-Order Critical Points in Vector Fields

There are numerous techniques that have been developed for the visualization of 3D vector fields. A thorough overview of vector field visualization techniques can for instance be found in [13, 23].

Vector field visualization techniques can be divided into two groups: geometry-based and texture-based. As the names suggest, geometry-based techniques employ geometric objects for visualization, while texture-based techniques use textures. However, this is not an exact split into two groups as certain types of techniques combine elements from both groups.

An example of a geometric technique is the use of glyphs. Glyphs are small icons spread throughout the vector field. Each glyph usually depicts the orientation and magnitude of the vector field at the location of the glyph, for example using an arrow where the arrowhead points in the direction of the vector field, and the length of the arrowhead depicts the magnitude. This technique is best suited for small vector fields. For large vector fields, such a visualization tends to be cluttered due to the large amount of glyphs required. Thus, the usefulness of this method is limited to small vector fields.

Another geometric technique is to visualize the vector field by drawing field lines. A number of seed points are selected throughout the vector field. Seed points can be selected at random, or better, according to a specific seeding strategy. The visualization of field lines produces a sparse representation of a vector field. Therefore, the quality of the visualization is highly dependent on the placement of the field lines. Typically, since an increasing number of field lines will lead to more clutter, a possible strategy is to select seed points close to the parts of the field which is most interesting according to the phenomena which is being studied.

Seeding strategies can be grouped into different classes depending on what metric they use to distribute the field lines. Strategies include density-guided seeding [41, 19, 27, 9], flow-guided seeding [42, 46], and more recently, similarity-guided field line placement [6]. Density-guided seeding is a seeding strategy that aims to reduce visual clutter by distributing field lines evenly in space. This is generally performed by inserting new field lines away from existing field lines, and terminating the field line integration if the line gets too close to a previously generated field line. The second approach, flow-guided seeding, tries to capture important vector field structures by seeding in regions of most interest first, such as the regions close to critical points or other interesting features. Similarity-guided field line placement is based on a metric called similarity distance. This metric not only accounts for the closest distance between two field lines, but also for their similarity of shape and direction.

A popular technique within the group of texture-based techniques is line integral convolution
Visualization of First-Order Critical Points in Vector Fields

(LIC) [4, 37]. LIC uses an input noise texture which is convolved with a filter kernel function in the direction of the vector field. This produces an image which is correlated along the field lines, but with no or less correlation in the perpendicular direction. For 3D visualizations of vector fields using LIC, volume rendering can be used to display the texture [33, 15].

Texture-based methods have a benefit over geometry-based techniques by not being dependent on the choice of seeding strategies. A texture-based method such as LIC will depict all parts of the vector field. Thus, it is not susceptible of missing important features of the data being visualized. Still, 3D LIC methods provide challenges. For instance, the computational cost for generating a full LIC texture has a cubic increase with respect to grid resolution. Also, a dense representation of the vector field in the form of a 3D LIC texture is difficult to visualize due to occlusion problems. An important limitation of LIC textures is that they do not convey information about the direction of the vector field along the field lines.

In this thesis, the focus is on the visualization of vector field topology. A suitable method should be able to clearly reveal the location of critical points, and also be able to visualize the local topological structure of the vector field around the critical point. To obtain such a visualization, we present three different techniques. The first is field line visualization with an adapted seeding strategy for capturing the topological structure around a critical point. The second technique is a variant of LIC that we have developed which will compute a 2D LIC texture of the tangential component of a vector field on a plane inside a 3D volume. The final technique we present is a hybrid approach that combines the two techniques into a single visualization.

An overview of topology-based flow visualization can be found in [22].

In addition to being able to clearly depict the topological structure of vector fields, it is important that the visualizations can be explored interactively. The parallel nature of the methods we present in this chapter are good candidates for being implemented as shader programs executed by a GPU (graphics processing unit). A modern GPU is in essence a SIMD (single instruction, multiple data) processor, and is able to do processing at a rate sufficient for interactive display of the visualizations.

5.1 Templated Seeding Strategy for Critical Points

To capture the field line patterns around critical points it is important to use a field line seeding strategy that will help illustrate the structure of the vector field in the vicinity of the critical point. Such a seeding strategy can also depict the vector field in a region outside the local region of the critical point so the viewer can get an understanding of the global structures of the field.

It is not given that a general seeding strategy alone will depict the location and type of critical points, especially when seeding sparse. This motivates the use of a templated seeding strategy in which field lines are seeded close to the critical points. What is important to this type of seeding strategy is that it is based on derived data, critical points, and a preprocessing step where the location and type(s) of critical points must be extracted from the vector field as a preprocessing step.

A technique that will help capture the details of critical points in 3D is presented in a work by Ye, Kao and Pang [46]. Their approach is to analyze each of the critical points, and determine a template for the seeding of field lines based on the type of critical point. The type of critical point is determined using the eigenvalues of the Jacobian matrix. Classification is simplified as compared to the detailed analysis given in [7], and in chapter 4 of this thesis. The reason for this simplification is to be able to morph between different seeding templates. Sometimes, the type of critical point will be close to changing into a different type of critical point, and the use of an adapted seeding template should be advantageous in terms of capturing the vector field.
topology around the critical point.

A similar work has been performed by Verma et al. for 2D vector fields [42]. They present a series of seeding templates for critical points. In addition, they have developed a heuristic for deciding the size of the seeding templates using Voronoi partitions. The use of Voronoi partitions allows them to precisely determine a region around a critical point in which the vector field behaves as if there were no other critical points present. They also use the partitions to determine the orientation of the seeding template.

The general procedure for a templated seeding strategy is to first extract the critical points, then classify them according to the properties of the Jacobian matrix, and finally to determine a suitable template for seeding around the critical point. An optional global seeding step can be performed last to help visualize the global structures of the vector field.

In the subsequent sections we present a seeding strategy that is based on the theory on classification of critical points given in chapter 4. This strategy will be focusing on the topology of the individual phase planes of critical points.

5.1.1 Template Scaling

A critical point in a vector field has a local region of influence were its vector field pattern is dominant. The templates described above can be directly applied to vector fields with a single critical point. For vector fields with multiple critical points, there is no immediate way of determining how to scale the seeding template. If two or more critical points are clustered together, chances are the seeding templates will intersect and reduce the quality of the visualization. Therefore, seeding templates should take into account the distance to nearby critical points.

There are multiple metrics for determining the region of influence for a critical point. In the introduction, we discussed Voronoi partitions. These partitions tessellate the vector field into non-overlapping regions, one for each critical point, where the vector field behaves as if no other critical points were present in the field. Computing a full Voronoi tessellation for the critical points is an expensive operation in terms of the computational effort required, and not required for our purposes, as we will see that simpler methods yield good results in general.

A very simple partitioning method is to simply divide the full volume of the vector field by the number of critical points to get an idea of the average influence region for each critical point. For this method there is an underlying assumption that the critical points are distributed uniformly throughout the vector field, which is seldom the case. A partitioning method must adapt the regions of closely spaced critical points so that there is no overlap.

One method that will guarantee that there is no overlap between the regions is to estimate the size of the seeding template for a critical point by computing the distance $D_{\text{near}}$ to the nearest critical point. By letting each critical point be surrounded by a spherical region with radius $D_{\text{scale}} = D_{\text{near}}/2$, none of the regions will overlap, thus creating distinct partitions. We use this scale as a guide when defining the phase plane seeding templates below.

5.1.2 Phase Plane Templates

In chapter 4, we showed that the characteristic phase planes of a critical point determine the type of vector field topology that surrounds the critical point. Therefore, a seeding scheme should be constructed so that it will capture the vector field topology in the phase planes. In table 4.2 on page 48, a complete classification scheme is given. This table shows how the combination of eigenvalues determine the type of phase planes for a critical point. It is an explicit classification that assigns to each of the phase planes of a critical point a specific two-dimensional topology, such as a node or a saddle. These two-dimensional topologies, when combined, make up a
three-dimensional topology for a critical point. This motivates a seeding strategy based on the two-dimensional topology of the individual phase planes.

There are similarities between the types of two-dimensional phase plane topologies. For instance, a center is a special case of a focus with no movement in the radial direction. Both the center and the focus topologies are results of the Jacobian matrix evaluated at the critical point having complex eigenvalues. The radial movement around such a critical point is determined by the magnitude of the real part of the complex eigenvalues. Thus, if the real part is zero, there will be no radial movement, and the focus topology is degenerated into a center topology. A similar transition can occur for nodes and star nodes. A phase plane has node or star node topology if the two eigenvalues corresponding to the two real eigenvalues, spanning the phase plane, are both positive or negative. If the values of the eigenvalues are identical, the topology degenerates from a node into a star node topology. These types of topology transitions for the phase planes can be incorporated into the seeding scheme using a template that adapts its structure according to the eigenvalues.

In the following sections, we will describe the different seeding templates for the two-dimensional phase plane topologies. We observe that all phase plane topologies have both an attracting and a repelling variant, for instance an attracting or repelling node. However, since field lines can be integrated both forward and backwards, there is no need to separate between the attracting and repelling variants of a given phase plane topology. When the terms eigenvector and eigenvalue are used below, they refer to the eigenvectors and eigenvalues, which can be complex, of a phase plane with the given topology. Some results using this templated seeding strategy is shown in figure 5.5 on page 57.

**Nodes and Star Nodes**

Nodes have a topology as shown in figure 5.1. As stated previously, the star node topology is a special occurrence of the node topology in which the eigenvalues are equal. For the star node topology, the trajectories are either directly into, or away from the critical point. For the node topology, the trajectories either emerge from, or move into, the critical point along one of the eigenvectors, but are parallel to the other eigenvector further away from the critical point.

A possible template for the node and star node topology is an ellipse. An ellipse has two parameters that can be controlled: the length of the semimajor axis, $a$, and the length of the semiminor axis, $b$, which occur in the equation for an ellipse centered around the origin in Cartesian coordinates:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1. \quad (5.1)$$

The length of the semimajor and semiminor axis can be used to control the eccentricity of the ellipse. Consider the eigenvalues for the phase plane containing the node or star node topology. By letting the eccentricity of the ellipse be controlled by the eigenvalues, the shape of the ellipse can be transformed so that the elongation is in the direction of the smaller of the two eigenvalues. This will make the ellipse elongated in the direction of the phase plane trajectories emerging from the critical point. By seeding along the ellipse, as shown in figure 5.1, this template should capture the characteristic topology of a node. If the two eigenvalues are equal, the ellipse is simply a circle, which is a suitable template for star nodes.

Assume that the eigenvalues are sorted so that $\lambda_1 \leq \lambda_2$. In case $\lambda_1$ and $\lambda_2$ are both negative, use the modified eigenvalues $\lambda_1^* = -\lambda_2$ and $\lambda_2^* = -\lambda_1$, and the corresponding modified eigenvectors $v_1^* = -v_2$ and $v_2^* = -v_1$. We define an elongation factor

$$c = \frac{\lambda_1}{\lambda_2}, \quad c \in (0, 1]. \quad (5.2)$$
Let \( a = D_{\text{scale}} \), and \( b = c \cdot D_{\text{scale}} \), where \( D_{\text{scale}} \) is the scaling factor defined in section 5.1.1. Define a new coordinate system with the two normalized eigenvectors \( v_1 \) and \( v_2 \), corresponding to \( \lambda_1 \) and \( \lambda_2 \), as basis vectors. These basis vectors are to correspond to the semimajor and semiminor axes of an ellipse given in the \((v_1, v_2)\) coordinate system. An ellipse in the new coordinate system is given by

\[
\frac{r^2}{a^2} + \frac{s^2}{b^2} = 1. \tag{5.3}
\]

A benefit of defining the ellipse in the coordinate system given by the eigenvectors, is that the ellipse will deform according to the orientation of the eigenvectors. We use equation (5.3) to compute seeding points along the ellipse using the following equation:

\[
s = \pm b \sqrt{1 - \frac{r^2}{a^2}}, \quad r \in [-a, a]. \tag{5.4}
\]

Define a discrete set of \( r \in [-a, a] \) into \( n + 1 \) uniformly spaced points:

\[
\Delta r = \frac{2a}{n}, \quad r_i = i \Delta r, \quad i = 0, 1, \ldots, n.
\]

We use this discretization to generate two sets of seeding points, \( p_1(r_i) \) and \( p_2(r_i) \) using

\[
p_1(r_i) = (r_i - a)v_1 + b\sqrt{1 - \frac{(r_i - a)^2}{a^2}}v_2, \quad \text{for} \quad i = 0, 1, \ldots, n \tag{5.5}
\]

and

\[
p_1(r_i) = (r_i - a)v_1 - b\sqrt{1 - \frac{(r_i - a)^2}{a^2}}v_2, \quad \text{for} \quad i = 0, 1, \ldots, n. \tag{5.6}
\]

These two equations generate a discrete representation of two half-ellipses. At the two points where two half-ellipses meet, there will be duplicate seeding points. To avoid seeding twice at the same location, a simple fix is to limit the interval for \( i \) for both of the sets to \([1, n-1]\). With this limitation, the scheme will generate a total of \( 2(n-1) \) seeding points on the ellipse.

### Logarithmic Nodes

Logarithmic nodes are similar to nodes and star nodes. The trajectories in the logarithmic node topology either emerge from, or move into the critical point in a direction that is parallel to
the eigenvector. A trajectory emerging from the critical point will start in a direction parallel to the eigenvector, and then turn and move in the opposite direction, creating the topology shown in figure 5.2. We use a seeding template where the seed points are placed on two lines parallel to the real eigenvector. The vector field is evaluated in one of the quadrants of the coordinate system defined by the two eigenvectors to determine in which two of the quadrants the characteristic topology resides.

**Focus and Centers**

Focus and centers appear in the single phase plane of a critical point whose Jacobian matrix has complex eigenvalues. These topologies have spiraling, or swirling, trajectories centered around the critical point. The phase plane is spanned by the real and imaginary vectors of the pair of complex eigenvectors. Let $\lambda = a + bi$ be the pair of complex eigenvalues. For focus and center topologies, the radial movement of the trajectories is determined by the value of $a$. For $a > 0$, the trajectories repel from the critical point, while for $a < 0$ the trajectories are attracted towards the critical point. Thus, if $a = 0$, there is no movement towards or away from the critical point, and this is the characteristic of a center topology.

For the center topology, we seed along a line in the phase plane with uniform spacing between the seed points (figure 5.3). This template, if applied to a focus, might fail to capture the spiraling structure if the divergence rate of the trajectories away from the critical points is high. Therefore, for focus topologies, we use a template with two seed points on a line passing through the critical point. One seed point is placed on the line on either side of the critical point, with equal distance to the critical point.

**Saddles**

Saddle points have a topological structure where the lines given by the eigenvectors of the Jacobian act as asymptotes for the field lines. Along the asymptotes, the direction of the vector field is either directly towards or away from the critical point, and field lines will cluster around these asymptotes. Outside the asymptotic regions, the field lines have a parabolic structure. It is essential to capture the asymptotic and parabolic structure of the vector field around saddle points, so the seeding template is constructed by seeding along the four half-angles that can be computed from the two eigenvectors as shown figure 5.4. The number of seed points along each of the lines bisecting the two eigenvectors can be varied according to $D_{\text{scale}}$ defined in section.
5.1.1 For example, if $D_{\text{scale}}$ is small, the number of seed points can be reduced to prevent clutter in regions of the vector field where the distance between critical points is small.

5.1.3 Eigenvector Seeding for Spirals

Critical points with phase planes of center or focus topologies suggest the existence of 3D spiraling structures. These critical points have complex eigenvalues, and therefore a single real eigenvector. To help capture these structures, if the critical point is part of such a structure, we seed in a point slightly offset from the critical point in the direction of the real eigenvector. In figure 5.6, we see that this eigenvector seeding is successful in capturing the center of the spiraling structure in the middle-right part of the image. This spiraling structure encompasses three critical points. A spiral center is also visible in the middle image of figure 5.5.

5.1.4 Critical Point Symbols

We visualize the exact location of critical points using colored spheres, where the color encodes the classification of the critical point. This will ensure that even if the seeding strategy fails to
Figure 5.5: Results using the various phase-plane seeding templates for 3D critical points. From left to right: node, focus and node saddle. The node has three phase planes with node structure. The focus has a spiraling structure in one phase plane. The node saddle critical point has a saddle topology in two of the phase planes, and a node topology in the third.

depict a critical point, the fact that a critical point exists is guaranteed to be visualized. Using the colored spheres alone helps the viewer understand in which regions the density of critical points is high, and the relative locations of the critical points. See figure 5.6 for an example of an illuminated field lines visualization where the location of critical points is marked using colored spheres.

5.1.5 Additional Global Seeding

An additional seeding process can be performed after the templated seeding to help depict the global structures of the vector field. The intent of this process is to generate additional field lines in regions not influenced by the critical points. We use a uniform random seeding in which the vector field size determines the number of such additional seed points. The strategy is to sample a uniform random distribution to get a random point within the vector field. This point is a candidate point. If the point is closer than some predetermined distance to a critical point, or is too close to a previously globally seeded point, it will be rejected and a new candidate point is generated. This process continues until a given number of globally seeded points have been generated. These additional field lines are given a low alpha value to help separate them from the field lines generated from the seeding templates of critical points. See figure 5.7 for an example.

5.2 Illuminated Field Lines Using GPU Vertex Shader

A simple 3D vector field visualization might display the field lines using flat shading of line segments. Often, color is used to encode the velocity of the field, or another scalar property. However, the use of flat shading does not give a good spatial impression of the field lines. A simple workaround is to generate polygonal tubes rather than simple connected line segments, and use a standard lighting model provided by the graphics API for shading. However, using polygons will reduce and limit the number of field lines that can be displayed.

Traditionally, real-time computer graphics has used Phong’s lighting model since it is easy to implement and fast to evaluate. However, this lighting model is based on the computation of normal vectors for flat surfaces. For a line segment there is infinitely many normal vectors orthogonal to the line. Because of this ambiguity, there is no straightforward way to apply the lighting model to line segments.
Figure 5.6: Color-coded spheres are used to show the location of critical points. Red spheres are critical points having complex eigenvalues. Blue spheres represent critical points with real eigenvalues. The dataset visualized is from [45].

Figure 5.7: Visualization with global seeding enabled. The dataset visualized is from [45].
In 1997, Stalling, Zöckler and Hege presented a technique that employs texture look-ups for calculating the light intensity of field lines [38]. This technique avoids the ambiguity problem for the normal vector, and a large amount of field lines can be displayed using a lighting model that does not necessitate the use of polygonal primitives.

Below we give a short introduction to Phong’s lighting model, and then we describe the original technique and how it can be implemented to run efficiently on a modern GPU as a vertex shader program.

5.2.1 Phong’s Lighting Model

The most widely used lighting model in real-time computer graphics is Phong’s lighting model which describes the interaction between light sources and object surfaces in a scene. Phong’s lighting model relies on normal vectors to describe the orientation of surfaces relative to light sources. At a given point on the surface, let \( \mathbf{N} \) be the unit normal vector, and \( \mathbf{L} \) be a unit vector in the direction of a light source from the surface. See figure 5.8.

The model divides the total light intensity at a surface point into 3 different types of contributions: ambient, diffuse and specular. The ambient component is the light that hits an object indirectly from other light sources, and is not accounted for in the other two contributions. It can be regarded as a type of constant background light coming from all directions. This component is independent of both the light vector and the normal vector. An object rendered using only this component will thus appear to be flat since every point receives the same contribution.

The second contribution is the diffusive light. This contribution stems from Lambert’s law which states that for surfaces that are ideally diffuse, i.e. without shininess, the reflected light is determined by the cosine of the angle between \( \mathbf{N} \) and \( \mathbf{L} \). If we call this angle \( \theta \), then we have \( \cos(\theta) = \mathbf{L} \cdot \mathbf{N} \), so the contribution can be computed using an inner product. A notable property of this component is that surfaces facing away from the light will not receive a diffuse contribution since \( \theta > \pi/2 \) for those surfaces. Negative values of \( \mathbf{L} \cdot \mathbf{N} \) are clamped to zero to avoid negative contribution. Using the diffusive contribution it is possible to determine the curvature of a rendered object. It should also be noted that the diffuse component is view-independent which means that a given lit point on a surface will look the same independent of viewing direction.

The third and final contribution is the specular component. This component takes into account the viewing direction and is the one that will generate shiny highlights when rendering materials such as metals and glass. Such highlights help to determine the curvature of objects, but also the location of light sources. To compute this model the reflection vector must be determined. This is simply the light vector reflected around the normal for perfectly reflective, flat surfaces. Let \( \mathbf{R} \) be the unit reflection vector, \( \mathbf{V} \) be the unit vector pointing from the surface towards the viewing direction and \( \rho \) be the angle between \( \mathbf{R} \) and \( \mathbf{V} \). The specular contribution is then given by \( (\cos(\rho))^n = (\mathbf{R} \cdot \mathbf{V})^n \) where \( n \) is the shininess factor. Again, if the resulting value is negative, it will be clamped to zero to prevent a negative contribution.

We can now calculate the total intensity of the light by adding the ambient, diffuse and specular terms according to the following equation:

\[
I = I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}}
= k_a + k_d(\mathbf{L} \cdot \mathbf{N}) + k_s(\mathbf{V} \cdot \mathbf{R})^n
\]

where \( k_a, k_d, k_s \) are the intensity coefficients.
5.2.2 Description of the Technique

Equation (5.7) is suitable when the normal vector is given or can be computed. But for line primitives there is an infinite number of vectors orthogonal to the tangent direction of the line, so we are unable to select a unique normal vector for each primitive. Instead, one can select the one which is coplanar to the light vector $\mathbf{L}$ and the tangent vector for the line primitive $\mathbf{T}$. This makes it possible to express the diffuse intensity contribution without using a specific normal vector. The light vector is decomposed into the normal and tangent spaces as shown in figure 5.9. This gives an orthogonal decomposition $\mathbf{L} = \mathbf{L}_N + \mathbf{L}_T$ where $\mathbf{L}_N = (\mathbf{L} \cdot \mathbf{N})\mathbf{N}$ and $\mathbf{L}_T = (\mathbf{L} \cdot \mathbf{T})\mathbf{T}$ are the projections. Applying Pythagoras’s theorem gives

$$\mathbf{L} \cdot \mathbf{N} = |\mathbf{L}_N| = \sqrt{1 - |\mathbf{L}_T|^2} = \sqrt{1 - (\mathbf{L} \cdot \mathbf{T})^2}. \tag{5.8}$$

This is a formula for computing the diffuse component that does not rely on the normal vector. A similar technique can be used for the specular term. We start by observing that $\mathbf{R}_N = \mathbf{L}_N$ and $\mathbf{R}_T = -\mathbf{L}_T$. The inner product $\mathbf{V} \cdot \mathbf{R}$ can be rewritten as:

$$\mathbf{V} \cdot \mathbf{R} = \mathbf{V} \cdot (\mathbf{L}_N - \mathbf{L}_T)
= \mathbf{V} \cdot ((\mathbf{L} \cdot \mathbf{N})\mathbf{N} - (\mathbf{L} \cdot \mathbf{T})\mathbf{T})
= (\mathbf{L} \cdot \mathbf{N})(\mathbf{V} \cdot \mathbf{N}) - (\mathbf{L} \cdot \mathbf{T})(\mathbf{V} \cdot \mathbf{T})
= \sqrt{1 - (\mathbf{L} \cdot \mathbf{T})^2}\sqrt{1 - (\mathbf{V} \cdot \mathbf{T})^2} - (\mathbf{L} \cdot \mathbf{T})(\mathbf{V} \cdot \mathbf{T}). \tag{5.9}$$

A similar expression to that of (5.8) has been used to rewrite $\mathbf{V} \cdot \mathbf{N}$. Formulas (5.8) and (5.9) can then be use in formula (5.7) to give a lighting model for a line segment. This lighting model does not require an explicit normal vector to be defined for the line segment.

One notable issue of using formula (5.7) for computing the light is the artificially excess brightness produced by the diffuse contribution. The normal vector is always selected so that it minimizes the angle between the light vector and the normal, resulting in an overall increased light intensity on the field lines. Stalling et al. [38] suggest using an exponent on the diffuse contribution term to reduce the excess brightness.

---

Figure 5.8: The unit direction vectors used in Phong’s lighting model.
Figure 5.9: Left: The relationship between the light vector $\mathbf{L}$ and the reflection vector $\mathbf{R}$. Right: Projection of the light vector into normal and tangent space. Figure is from [38].

5.2.3 Alpha Masking for Critical Points

The visualizations of fields with a large number of critical points will contain a large number of field lines using the templated seeding strategy. To emphasize the location of the critical points, one can use alpha channel masking to help the viewer discern which field lines are closer to the critical points. Also, field lines that are generated in the global seeding step (section 5.1.5) are masked since these field lines are less important for understanding the topology of the vector field. For field lines seeded from critical points we use the formula

$$\alpha = 1 - \frac{d}{L},$$

where $d$ is the distance from the nearest critical point, and $L$ is a constant that decides how far away from the critical point the field line will obtain maximum transparency. To avoid having field lines become completely transparent, we clamp values that are below a certain threshold $\kappa$ so that the alpha value assigned to the line segment lies in the interval $[\kappa, 1]$. Field lines introduced by the optional global seeding step are given a constant low alpha value to help discern them from the field lines seeded from critical points.

5.2.4 Depth Attenuation

A useful addition to the visualization of field lines, is to help the viewer understand the depth location of field lines by using depth attenuation to darken field lines that are far away from the view. This is especially useful when visualizing vector fields with a high number of critical points, since the number of field lines can be large, thus possibly increasing the amount of overlapping field lines.

Field lines that are further away from the light source should be attenuated more than field lines that are close to the light source. The attenuation formula we use is that typically found in a graphics API. The attenuation is given by

$$A(d) = \frac{1}{C_a + L_ad + Q_ad^2},$$

where $A$ is the attenuation factor, $d$ is the distance from the light source, and $C_a, L_a$ and $Q_a$ are the constant, linear and quadratic attenuation terms, respectively.
Figure 5.10: Flat shading using the ambient intensity. It is difficult to perceive the spatial location of the field lines.

Figure 5.11: Ambient and diffuse shading. This shading helps the perception of depth in the image.
To avoid field lines from becoming invisible, the depth attenuation is only applied to the diffuse and specular components. The modified Phong’s formula that takes the depth attenuation into account is

\[ I = I_{\text{ambient}} + A(I_{\text{diffuse}} + I_{\text{specular}}) \]  

where \( k_a, k_d, k_s \) are the intensity coefficients, and \( A \) is the depth attenuation factor. See figure 5.13 for an example of the effect produced by depth attenuation.

### 5.2.5 Vertex Shader Implementation Details

The visualization software was developed using the OpenGL graphics API in combination with NVIDIA’s Cg language for the vertex shader. As mentioned in the introduction of this chapter, the original article specifies textures as look-up tables to compute the diffuse and specular

Figure 5.13: This figure shows the effect of depth attenuation. Depth attenuation is applied to the light source in the image on the right.
Visualization of First-Order Critical Points in Vector Fields

intensity contributions given the tangent vector. This is no longer required since programmable GPUs can compute the modified lighting model as part of the vertex processing in the graphics pipeline.

Visualization of large vector fields with many critical points require a large amount of field lines to be generated. To increase the performance, we use arrays to store vertex coordinates, tangent vectors and color information. The renderer receives these arrays using a few function calls as opposed to having to send a single vertex with tangent vector and color information at a time.

Several coordinate systems are involved in the OpenGL graphics pipeline. In OpenGL, objects that are to be rendered are first defined in an object coordinate system which is transformed to eye coordinates by multiplication with a modelview matrix $M$. The modelview matrix is the combination of modelling and viewing transformations. Often, the eye coordinate system is defined as a world coordinate system since the modelview transformation can be regarded as scaling and rotating objects, and then placing them in the scene by a translation. The viewer is placed at the origin. It is natural to use the eye coordinate system when defining the location of the light source. Since the viewer is located at the origin, the light source can be placed relative to the viewer. This way the light source will follow the viewer as he is studying the visualization. Also, computing the specular light contribution is easier to do in eye coordinates, else an inverse transformation to determine the location of the light source in object coordinates is required.

It is important that the direction vectors from the line segment vertices to the light source and viewer are given in the same coordinate system. The position and tangent of the line segment vertex is therefore transformed to eye coordinates before the unit direction vectors and light intensity coefficients are computed.

The full Cg source code for the illuminated field lines vertex shader program is available in figure 5.14. For examples of visualizations using various combinations of lighting parameters, see figures 5.10 to 5.12.
void vertexMain(float4 position : POSITION,
    float3 tangent : NORMAL,
    float4 color : COLOR,

    out float4 outPosition : POSITION,
    out float4 outColor : COLOR,

    uniform float3 viewPos,
    uniform float3 lightPos,
    uniform float Ka,
    uniform float Kd,
    uniform float Ks,

    uniform float Ca,
    uniform float La,
    uniform float Qa,

    uniform float4x4 modelView,
    uniform float4x4 modelViewProj)
{
    outPosition = mul(modelViewProj, position);
    tangent = normalize(tangent);

    float3 eyePos = (float3)mul(modelView, position);
    float3 eyeTangent = mul((float3x3)modelView, tangent);
    float3 Lnn = lightPos - eyePos;

    float3 T = normalize(eyeTangent);
    float3 V = normalize(viewPos - eyePos);
    float3 L = normalize(Lnn);
    float LT = dot(L, T);
    float VT = dot(V, T);
    float VR = max(sqrt(1.0 - LT*LT)*sqrt(1.0 - VT*VT) - LT*VT, 0);

    float ambient = Ka;
    float diffuse = Kd*pow(sqrt(1.0 - LT*LT), 4);
    float specular = Ks*pow(VR, 40);

    float d = length(Lnn);
    float att = 1.0/(Ca + (La + Qa*d)*d);
    float sum = ambient + att*(diffuse + specular);

    outColor.rgb = color.rgb * sum;
    outColor.a = color.a;
}

Figure 5.14: Cg source code for the illuminated field lines vertex shader program.
5.3 Tangent LIC for Critical Point Visualization

In chapter 4, we showed that the characteristic vector field topology around a critical point is described by the phase portraits in specific planes. This motivates a visualization strategy for planes slicing through a 3D vector field.

The criteria for a good critical point visualization are as follows: depict the vector field in the phase planes and provide a high level of detail in a small domain centered around the critical point. What we need is a technique that is able to balance the competing requirements of depicting small-scale structures close to critical points, while not creating a cluttered visualization.

The strength of geometric field line visualization lies in its ability to depict the global 3D structures of the vector field, and the interactions between different topological structures. However, small-scale features can easily be missed because there is no a priori knowledge of how densely field lines must be placed.

For a two-dimensional vector field, line integral convolution would meet the given criteria. This motivates an investigation on the use of three-dimensional LIC visualization techniques. The technique is easily derived from 2D LIC by adding a third dimension to the input noise texture, and modifying the integration so it can be used for 3D vector fields. The output from this method is a 3D texture which is typically visualized as a voxel set using a volume renderer. This technique meets the criteria of providing a high level of detail, but there is no immediate way of extracting the phase portraits of the critical points.

If a polygon is defined in such way that it slices a 3D LIC texture with the same position and orientation as the phase plane, and the texture is mapped on to the polygon using texture mapping, an image of the phase plane is produced. If the vector field orientation in the slicing plane is everywhere tangent to the plane, the image will be correlated along the field lines. This is seldom the case. Visualizing a single slice of an opaque 3D LIC texture, will in general produce uncorrelated images. In figure 5.16 such a texture is visualized using volume rendering. Even though volume rendering is used, the texture is opaque and the slicing plane depicts only the texels actually sliced. The result is large uncorrelated areas where directional information is absent. To explain the problem in more detail we define two sets of voxels. The first set is the set of voxels that intersects the slicing plane. The second set is the set of voxels intersected by a given field line. The amount of correlation is dependent on the intersection between the two voxel sets. A greater intersection between the two voxel sets will result in more correlation. An illustration to the problem is provided in figure 5.15.

Ways of visualizing 3D LIC textures and creating textures suited to depict its internal structures have been developed [33, 15]. SeedLIC [15], for example, produces a sparse texture which makes it easier to create a correlated image by volume rendering in such a way that not only the texels sliced by a clip plane, but also layers underneath are visible. This effect requires manipulation of a look-up table. By performing the same manipulation while using dense textures, as the one in figure 5.16, one can change the opacity of certain ranges of the grey scale voxels, thus creating transparency. This will increase the correlation as shown in figure 5.17. As for SeedLIC, some of the visible structures are not actually in the plane, and the viewer are no longer looking at a specific plane, but a small volume. A moveable clipping plane operated by the user, or in an animation, will also increase the perception of the structures close to the plane. In our case this is not a good choice for obvious reasons. One being the need of maintaining the exact position and orientation of the plane, to depict the correct phase portrait.

A part from the unsatisfactory correlation in the produced images, the fact that they are produced using some kind of volume rendering technique, makes this approach unsuited for our application. We want to visualize a polygon with a LIC image on it, as if simple texture mapping
Figure 5.15: The correlation between the voxel set sliced by a plane and the voxels encoding a LIC field line. The voxels rendered in green are intersected by both and represent the union of the two voxel sets. The voxels rendered in red are only intersected by the plane. These will cause loss of correlation when rendering the total voxel set using volume rendering and texture mapping.

was used, but with the correlation of a 2D LIC image. To work around the correlation problem we created a new LIC routine for computing a 2D LIC texture for an arbitrary plane slicing a vector field. The idea is to constrain the directional information by only tracing the tangential components of the vector field. LIC has previously been applied to surfaces in three dimensions [3, 26]. These methods do not compute the LIC images in real-time, and are dependent on visualizing a static surface geometry. In this thesis we present an interactive LIC visualization technique for moveable and scalable planes in three dimensions. To our knowledge, this has never been done before.

Limiting the visualization to only depict the tangential component of the vector field will limit the usefulness of this type of visualization for a general slicing plane. In the present context however, the planes are centered on critical points and oriented so that they correspond to the phase planes. In section 4.6 it was shown that the deviation of the actual flow direction from the tangential direction of the plane is zero in the phase planes, sufficiently close to the critical point. However, for a general vector field, the linearization which gives the system of ODEs is only valid within the cell of the critical point. So, for a general vector field, the actual field lines will deviate from the LIC plane. Even when centered on critical points and aligned with the phase planes as intended. When used in the context of visualizing the topology of critical points, this deviation from, or simplification of, the true vector field direction, is favourable if used correctly. To explain this we consider two geometrical objects. These two objects are topologically equal in shape if they can, without cutting or gluing, be transformed into the same shape. The flow around a critical point in a three dimensional flow field can be viewed as such an object. By using Tangent LIC in the phase planes, a simplifying transformation of the shape is performed, and the topological type of the planes are more easily determined.

5.3.1 Introduction to Line Integral Convolution

Line integral convolution is a texture-based visualization technique. The method is based on blurring a given noise texture along the field lines of the given vector field to generate an output texture. Such a technique itself does not reveal the magnitude, only the direction of the vector field. To depict magnitude, further information such as color must be added to the output image. What we will describe here will apply to both 2D and 3D variants of the technique. We
Figure 5.16: An opaque 3D LIC texture rendered using volume rendering. The green lines are the contours of a clip plane. Inside these contours only the texels sliced by the plane are visible.
Figure 5.17: A semi-transparent 3D LIC texture rendered using volume rendering. The green lines are the contours of a clip plane. The transparency helps to depict the internal structures.
will use the term texel to denote a texture element. This is in principle a pixel in an image for the 2D case.

The LIC algorithm has two sets of input data: the vector field and a noise texture, usually white noise. White noise is simply an independent random value generated for each texel from a uniform random number distribution. For each texture element, a line integral along the field line passing through the texture element is computed. This line integral is computed as a convolution between a given filter kernel function and the noise texture along the field line. The result from this operation is that along the field lines the value of the output texture is highly correlated in the tangential direction, while in the normal direction there is little or no correlation. This will produce an output image that clearly depicts the direction information of the vector field.

The mathematical formulation of a line integral convolution is

$$I(x) = \int_{s_0-L}^{s_0+L} k(s - s_0)T(r(s)) \, ds.$$  

Here, $I(x)$ is the output intensity of a texture element located at $x$. The input noise texture value at position $x$ is $T(x)$. The filter kernel function is given by $k$, and has length $2L$. The field line $r(s)$ is parametrized using the arclength $s$. This line integral is computed for each output texel, generating an output texture with the results. See figure 5.18 for an example.

5.3.2 Overview of the Tangent LIC Method

To visualize the field lines in a plane in a 3D vector field, we adapted Cabral’s 2D LIC routine to incorporate a change of basis from the standard Euclidean basis to the basis defined by the vectors spanning the slicing plane $P$. We need to generate two matrices which allow us to map the coordinates of a point, or the components of a vector, from the vector field physical coordinate system to the coordinate system spanned by the tangent vectors of $P$. We want to be able to express a direction, that is a vector from the vector field, in a point which lies on the plane, in the basis of the plane. This would allow us to disregard the component of the vector normal to the plane, and thus only move from point to point in the plane.

This can be compared to a drop of water on a windshield. The forces exerted by the wind on the drop are our vector field. In a simplified case, the forces of the wind are the only forces acting on the drop, and therefore the drop will move according to the pathlines of the field, or the direction of the wind at any instant. We know this is not the case, so we take one step towards
Visualization of First-Order Critical Points in Vector Fields

reality and add a force normal to the windshield, which are at all times equal but opposite to
the one caused by the wind, normal to the wind shield. That is, the drop can now move freely
on the windshield, but not lose contact with it. This is what we accomplish with our change
of basis and neglection of the normal component. We will now be able to create an image with
correlation, as long as the direction of the field lines is not entirely normal to the plane. In that
case, the vector we produce by removing the normal component will be the zero vector, and thus
no movement will occur, leaving us with an output intensity equal to the original white noise.

5.3.3 Coordinate Systems and Transforms

There are several coordinate systems involved in the Tangent LIC routine. In this section we
discuss the various systems, and how to transform between them.

The initial coordinate system is the underlying physical coordinate system of the vector field.
For structured, uniform grids, this vector field has a parametric representation where the vectors
are indexed in the range \([0, n_x - 1] \times [0, n_y - 1] \times [0, n_z - 1]\) for a vector field with dimensions
\((n_x, n_y, n_z)\).

This coordinate system can be mapped to a normalized representation so that 0 corresponds
to the first vector in each dimension, and 1 corresponds to the last. We will later show how
textures are used to hold the vector field, and a normalized representation will correspond with
texture coordinates. This normalized mapping is obtained by simply dividing each component
by the length of the corresponding dimension.

\[
x' = \frac{x}{n_x - 1}, \quad y' = \frac{y}{n_y - 1}, \quad z' = \frac{z}{n_z - 1}.
\]

The next coordinate system is defined by the basis given by two orthonormal vectors spanning
the slicing plane, and a third vector normal to the plane. The planes are defined by a point in
the plane, and the normal vector of the plane. The first vector in the basis is an arbitrary vector
\(a\) orthogonal to the normal \(n\) (figure 5.19). This is calculated by solving the equation of the
inner product so that \(a \cdot n = 0\). The second vector \(b\) should be orthogonal to both the normal
vector and \(a\), and is therefore defined by the vector cross product \(b = n \times a\). The vectors are
then normalized to create an orthonormal, right-handed basis.

The change of basis matrix from the plane basis to the standard basis is a \(3 \times 3\) matrix \(M\)
where the vectors of the plane basis is its columns, i.e.

\[
M = \begin{bmatrix}
  a_1 & b_1 & n_1 \\
  a_2 & b_2 & n_2 \\
  a_3 & b_3 & n_3 
\end{bmatrix},
\]

and the change of basis matrix from the standard basis to the plane basis is the inverse matrix
\(M^{-1}\).

The center of the slicing plane is defined to correspond to the critical point \(cp\) being visu-
alized. The four vertices of this plane are defined by linear combinations of the vectors \(a\) and
\(b\):

\[
fc_0 = cp - \frac{s}{2}(a + b) \quad \quad fc_2 = cp + \frac{s}{2}(a + b)
\]

\[
fc_1 = cp + \frac{s}{2}(a - b) \quad \quad fc_3 = cp - \frac{s}{2}(a - b),
\]

where \(s\) is the length of one side of the quadratic plane. The name \(fc\) is short for field coordinates,
and \(fc\) is the normalized vector field coordinates. Meaning that \(fc = (0,0,0)\) correspond to
Figure 5.19: The change of basis between the standard basis \((e_1, e_2, e_3)\) and the basis \((a, b, n)\) defined by the plane.

the lower left corner, \((0, 0, 0)\), of the vector field in physical coordinates, while \(fc = (1, 1, 1)\) correspond to the physical coordinates \((n_x - 1, n_y - 1, n_z - 1)\) of a vector field with dimensions \((n_x, n_y, n_z)\).

The two dimensional coordinate system of the plane, spanned by the vectors \(a\) and \(b\), is defined as follows.

\[
\begin{align*}
c_0 &= (0, 0) & c_2 &= (0, 1) \\
c_1 &= (1, 0) & c_3 &= (1, 1),
\end{align*}
\]

where \(c_0 \ldots c_3\) are the corners of the plane, i.e. the critical point and center of the plane \(cp\) will have the 2D plane coordinates \((0.5, 0.5)\), and \(c_0\) correspond to \(fc_0\) etc. Let \(c_k\) be the 2D coordinates of a point in the plane, and \(cp\) the 3D coordinates of the critical point. The 3D coordinates, \(fc_k\), of the point can be calculated in the following way

\[
\begin{align*}
v &= [(c_k - (0.5, 0.5)) \times 2, 0]^T \\
fc_k &= cp + Mv.
\end{align*}
\]

The 2D coordinates \(c\) will be used as the integration coordinates and for look-ups in a 2D texture holding the white noise. The 3D coordinates will be used to look-up the vector in a point \(c_k\) in the plane from a 3D texture holding the vector field. The latter will require a transform between the coordinate systems. Figure 5.20 explains the applicability of the two coordinate systems.
Figure 5.20: The two different coordinate systems. In the two dimensional one, the corners of the plane map to the corners of the 2D textures. In the three dimensional one, the corners of the vector field bounding box map to the corners of the 3D texture $T_V$ holding the vector field. All coordinates are normalized, and each component is only valid in $[0, 1]$. 
5.3.4 Tangent LIC Implementation

The Tangent LIC routine that we describe can be implemented as a standard application running on a CPU. However, LIC is a computationally intensive operation that cannot be performed at a rate sufficient for interactive visualizations with current generation CPUs. Therefore, we have developed the Tangent LIC routine so that it can be implemented in a fragment shader program running on a graphics processor.

The Tangent LIC routine computes field lines on a per-fragment basis. A fragment is a term used to describe a candidate pixel in the rendered image. Each fragment is processed by a fragment shader program which is executed on a GPU. The GPU will perform the computations in parallel, allowing the routine to be used in interactive visualization were a full LIC is performed for every rendered frame.

We use 3 textures for the computation. The first is the 3D vector field texture, $T_V$, containing the components of the vectors in the vector field. The details of how this texture is constructed is provided in section 5.3.5. Next, we have the 2D noise texture $T_N$. The resolution of this texture is determined by the resolution of the rendered image. Finally, a 2D texture $T_I$ is used for mapping between the coordinate system of the plane $P$ and the vector field, thus mapping from a 2D subspace to the Euclidean 3D space of the vector field.

As mentioned in section 5.3.3, a transformation between the 2D coordinates $c_k$ of a point in $P$ and the 3D coordinates $f_{c_k}$ has to be performed. Since the $P$ is rectangular, a bilinear interpolation between values defined at the corners $c_0, \ldots, c_3$ can be performed. If these values are the coordinates $f_{c_0}, \ldots, f_{c_3}$, the coordinates $f_{c_k}$ corresponding to an arbitrary point $c_k$ are computed. In the implementation of Tangent LIC, the coordinates of the vertices are stored in the $2 \times 2$ texture, $T_I$. This way, by performing a look-up in $T_I$ using the coordinates $c_k$, the interpolation is performed in graphics hardware with nearly no performance cost. Figure 5.21 explains how the coordinates are stored in the texture, while figure 5.22 shows how the different coordinate systems are used to retrieve information from the textures.

We will now proceed to describe the individual steps of the Tangent LIC method. The fragment shader is executed once for every fragment corresponding to a rasterized element of $P$. Each fragment is assigned texture coordinates $c_n$ by the graphics system. These coordinates are in the 2D coordinate system of $P$, and can be used directly to look up the corresponding grey scale intensity from the noise texture $T_N$ when needed. The initial position $c_k$ of the field line integration are these texture coordinates, corresponding to the fragment the shader is executed for. It is in the coordinate system of $P$ that integration will be performed. To obtain the tangential direction information from the vector field for every position in the 2D subspace, we need to map these coordinates to the 3D position in the vector field, and then retrieve the corresponding vector. To find the 3D coordinates corresponding to $c_k$, we perform a look-up in $T_I$ at coordinates $c_k$. The value returned is the 3D position $f_{c_k}$ in the vector field in normalized vector field coordinates.

Due to a linear transformation performed on the coordinates internally in the graphics pipeline, the position $f_{c_k}$ must be transformed so that the vectors of the underlying uniform grid are mapped correctly to the elements of $T_V$. This is described in section 5.3.5.

The velocity vector $v_k$ retrieved from $T_V$ is multiplied by $M^{-1}$ to transform the vector so it is expressed in the basis given by $(a, b, n)$. The normal component is set to zero to neglect the velocity in the normal direction of $P$, and the vector is normalized. The next integration position $c_{k+1}$ is computed by a simple fixed-length Euler step $c_{k+1} = c_k + h v_k$, where $h$ is the integration step length. A similar formula $c'_{k+1} = c'_k - h v_k$, where the direction of the vector field has been reversed, is used to generate a sequence of integration points $c'_k$ backwards along the field line.
This process is repeated for a given number of iterations. The field line integration is performed both forward and backwards from the initial position \( c_n \) in \( T_N \).

In each integration step all coordinates are checked for validity. The integration is stopped if any of the components of \( c_k \) or \( f c_k \) are outside the unit interval. The integration is also terminated if the magnitude of the velocity vector is below a certain threshold.

Given that the implementation performs \( n \) integration steps, the formula for the intensity of a output fragment \( f \) is

\[
I(f) = \begin{cases} 
0 & \text{if } n = 0 \\
\sum_k T(c_k) + \sum_k T(c'_k) & \text{if } n > 0. 
\end{cases}
\] (5.14)

A pseudo code for the algorithm is presented in figure 5.24.

### 5.3.5 Vector Field Texture

Since our algorithm will be running on a GPU, we need a way of representing the vector field components that is suitable for processing by the GPU. A natural approach is to use 3D textures since they support up to 4 components of data per texture element. The vector field components are represented in the color channels \((r, g, b)\), i.e., the \( x \)-component is mapped to \( r \), \( y \) to \( g \) and \( z \) to \( b \). We used 32-bit floating point textures provided by an OpenGL ARB extension. This is necessary for storing values outside the range \([0,1]\) in OpenGL textures.

A significant benefit of representing the vector field in a 3D texture is that the texturing hardware provides a function for performing trilinear interpolation. This makes vector field interpolation a computationally inexpensive process.

Textures are addressed using a parametric unit coordinate system independent of the resolution of the texture. In OpenGL, 1D textures are addressed so that 0 and 1 correspond to the left side of the leftmost texel and the right side of the rightmost texel, respectively. This applies to 2D and 3D textures as well for the additional dimensions. Consider the 2D texture shown in figure 5.23. Each of the 25 texels correspond to a vector in a 2D vector field. Since the vectors in the texture are be defined at the centers of each texel, and not on the vertices as in
Figure 5.22: The 2D texture coordinates, $c$, of a point $p$ in the plane are used to look-up the field coordinates, $fc$, in the texture $T_I$. These are used to look-up the corresponding vector stored in the vector field texture $T_V$. For the look-up of the grey scale intensity $I$ in the 2D noise texture, the 2D coordinates $c$ are used directly.
the vector field grid, we need to perform a linear transformation from the normalized vector field coordinates \((x', y', z')\) to texture coordinates. A general linear transformation \(f\) from \([a_1, b_1]\) to \([a_2, b_2]\) is given by

\[
    f(x) = a_2 + \left( \frac{b_2 - a_2}{b_1 - a_1} \right) (x - a_1).
\] (5.15)

Let \(\delta r\) and \(\delta s\) be defined by the width and length of a texel, respectively. We refer to these variables as offset factors. Assume the vector field coordinates have been normalized to the unit interval \([0, 1]\). We need to transform from \([0, 1]\) to \([\delta r/2, 1 - \delta r/2]\) for the \(x\)-component, and from \([0, 1]\) to \([\delta s/2, 1 - \delta s/2]\) for the \(y\)-component. The required transformations are found using formula (5.15):

\[
    T_1(x) = \frac{\delta r}{2} + (1 - \delta r)x
\] (5.16)

\[
    T_2(y) = \frac{\delta s}{2} + (1 - \delta s)y,
\] (5.17)

where \(x\) and \(y\) are the normalized vector field coordinates. For 3D textures the generalization is easily obtained by adding a third equation of similar form.

### 5.3.6 Fragment Shader Programming

Since real-time visualization is the target, we need an efficient way of computing the LIC planes for visualization. The Tangent LIC routine we have described is well-suited for parallel processing because the computation can be performed independently for each output fragment.

We implemented the LIC routine in a fragment shader program. This program is executed once for each fragment rasterized by the LIC plane. The input to the program is the two matrices \(M\) and \(M^{-1}\) needed for the change of basis, the normalized vector field coordinates of the vertices of the plane, and the offset factors \(\delta r, \delta s\) and \(\delta t\) needed for the linear transform from normalized vector field coordinates to texture coordinates. The output from the program is the color of the fragment being processed. The pseudo-code for the fragment shader program is given in figure 5.24.
FOR each fragment:
    pos2D = 2D noise texture coordinates of fragment;
    color = black;
    samples = 0;

    // Forwards integration
    WHILE samples < max_samples/2:
        IF invalid coordinates:
            BREAK;
        ENDIF
        color += textureLookup2D(pos2D, noiseTexture);
        samples++;
        pos3D = textureLookup2D(pos2D, pos3DTexture);
        vec = textureLookup3D(pos3D, fieldTexture);
        vec = matrixMultiply(basisChangeMatrix, vec);
        vec.z = 0;
        normalize(vec);
        pos2D += h*vec;
    ENDWHILE

    pos2D = 2D noise texture coordinates of fragment;

    // Backwards integration
    WHILE samples < max_samples:
        IF invalid coordinates:
            BREAK;
        ENDIF
        pos3D = textureLookup2D(pos2D, pos3DTexture);
        vec = textureLookup3D(pos3D, fieldTexture);
        vec = matrixMultiply(basisChangeMatrix, vec);
        vec.z = 0;
        normalize(vec);
        pos2D -= h*vec;
        color += textureLookup2D(pos2D, noiseTexture);
        samples++;
    ENDWHILE

    IF samples > 0:
        color /= samples;
    ENDIF
    RETURN color;
ENDFOR

Figure 5.24: Pseudo-code for Tangent LIC.
5.3.7 Textures and Memory Usage

Vector fields and noise data can be represented in texture memory in several ways. A 3D vector field can be stored in a 3D texture, or the vector field can be split up and marshalled into a set of 2D textures. The use of 3D textures is advantageous, since most graphics hardware can perform nearest-neighbor or trilinear interpolation of textures effectively. Also, using 2D textures increases the complexity because of the need to manage a potentially large set of textures. However, since the Tangent LIC algorithm in principle generates a synthetic texture on the slicing plane, it is more natural to think of the noise as being a 2D object.

It is possible to use 3D textures to represent the noise texture. In terms of implementation, a 3D texture representing the noise simplifies the set of coordinate transformation required since the texture coordinates of the vector field can be used to locate the texture element at the corresponding position in the noise texture without having to do a coordinate transformation.

Using a 3D texture allows the use of the same texture coordinates for both the vector field texture and the noise texture. This way, only the noise texels actually sliced by the plane is used in the computation of the intensity of the output pixel. The quality of the visualization is in large degree determined by the resolution of the noise texture, and a noise texture with dimensions larger than the field itself is in general needed. When increasing the dimensions of the 3D noise texture to meet this requirement, an even larger part of the texture memory is allocated, but never used in the computations.

Consider the following example: a slicing plane is normal to a standard basis axis, and the size is identical to that of the vector field in the orthogonal directions. Using the same texture coordinates for both the vector field and the noise texture results in the corners of the slicing plane being mapped to points on the boundaries of the noise texture. By modifying the resolution of the noise texture, the number voxels intersected by the plane can be controlled. The ideal size of the noise texture yields a number of intersected voxels equal to, or slightly less than the number of rasterized elements in the output image. Now, when the slicing plane is resized, for instance to half the original size in both directions, the number of intersected voxels decreases by a factor of 4. To compensate for the reduced intersections, the resolution of the noise texture must be increased correspondingly. Thus, reducing the size of the plane to half its original size requires 8 times as many texture elements in the noise texture.

Given a rendered image resolution of 800×800, and a slicing plane as described above covering the full extent in two dimensions of the field, the memory required is 488 MB. Decreasing the size of the plane to one tenth of its original size, the memory requirements increase to 476 GB. Given that typical modern consumer graphics cards have between 256 MB and 1 GB of graphics memory, this a requirement that is impossible to meet. The obvious solution is to use a lower texture resolution, but this will severely decrease the quality of the rendered image.

To overcome this we defined the noise texture to be a 2D texture with texture coordinates mapped so that

\[ c_0 \rightarrow (0,0), \quad c_1 \rightarrow (1,0), \quad c_2 \rightarrow (1,1), \quad c_3 \rightarrow (0,1). \]

This way, the mapping is one-to-one, and the number of texels in the noise texture intersected by the slicing plane is independent of the size of the plane. The use of a 2D texture requires an extra coordinate transformation for each integration step. However, this is computationally inexpensive when performed by a texture look-up as described in section 5.3.4.

5.3.8 Performance Considerations

For traditional LIC methods, the performance is dependent on the resolution of the output image and the length of the field line integrated. We assume that the integration step size is
included in the analysis of the length of the field line. The length of the field line is ideally some factor of the output image size. This dependency between the field line length and the number of pixels makes the complexity of the algorithm quadratic.

Tangent LIC is also quadratic, but the control of the parameters differs. Fragments are the analogy to pixels in the traditional methods, and affects the performance in the same way. The number of fragments is however controlled by the resolution of the rendering window, and the zoom and orientation of the view. The size of the textures used affects the quality of the rendered image, but not the performance.

5.3.9 Handling Aliasing Effects

Texture mapping can produce aliasing effects. This occurs when the sampling rate of the framebuffer is too low, resulting in multiple texels being covered by a single fragment. The LIC slicing planes, when viewed from distance, are susceptible of producing these aliasing effects.

To reduce aliasing we use mipmapping of the textures. Mipmapping generates a series of lower-resolution variants of the texture called mipmaps. When the image is rendered, the coverage ratio of the texture with respect to the framebuffer is used to decide which of the mipmaps to use for texturing. Thus, textured objects viewed at a distance will use a lower-resolution texture than objects closer to the view position. This results in reduced aliasing as can be seen in figure 5.25. For a detailed description of mipmapping, see [1].

5.3.10 Color Coding Field Line Direction

A limitation of LIC textures is that they do not convey information about the orientation of the vector field along field line structures. For instance, for a focus critical point topology, the LIC texture alone does not show if the focus is of a repelling or attracting type. This can be improved by using color to encode the orientation of the vector field along the field lines. Consider figure 5.26, where we have defined two points $p$ and $p_0$ in a LIC plane. Point $p_0$ is located at the center of the plane, i.e. it is the location of a critical point, and $p$ is the point for which we are to encode the orientation $v$ of the vector field. Vector $v$ is the normalized tangent component.
Figure 5.26: This figure shows the projection of the normalized tangential vector field component $v$ onto the line extending from the center of the plane $c$ to the point $p$, for which to calculate a direction color.

of the vector field relative to the plane. We define a new vector

$$u = \frac{p_0 - p}{|p_0 - p|},$$

(5.18)

which is a normalized vector from $p$ towards the center of the slicing plane. We can then use the inner product $u \cdot v = \cos (\theta)$ to compute the length of the projection of $v$ on the line extending from $p$ in the direction of the critical point. The length of this projection is a metric $r$ that we use to determine if the direction of the vector field is towards or away from the critical point in the slicing plane. Color is used to depict the orientation by letting $c_1$ be a vector of RGB color components representing flow towards the critical point, while a different color $c_2$ represents flow away from the critical point. We use the projection as a metric to linearly interpolate between the two colors $c_1$ and $c_2$. Since $\cos (\theta) \in [-1, 1]$, we apply a linear transformation from $[-1, 1]$ to the unit interval $[0, 1]$. The equation for computing the color depicting the orientation of the vector field along the field line relative to the center of the slicing plane, $c_{dir}$, is then given by the linear interpolation

$$c_{dir} = rc_1 + (1 - r)c_2,$$

(5.19)

where

$$r = \frac{u \cdot v + 1}{2}.$$

In figure 5.27, a LIC plane has been color-coded to show this extra orientational information.

### 5.4 Hybrid Visualization

We have seen two different techniques for visualizing the topology around critical points. In this section, we introduce a hybrid approach that combines the two techniques into a single visualization. The motivation for this is to be able to capture the details close to the critical point, and its phase plane topology using the Tangent LIC planes, while using field lines to give a good 3D visualization of the overall topology. The strength of Tangent LIC lies in the high level of detail in the LIC image, and its ability to simplify the topology to distinct planes. The primary weakness is its inability to depict the 3D motion of the vector field. Well-seeded illuminated
field lines visualize the 3D flow around a critical point well, but will not always capture the topology in a way that enables visual classification. By combining the two techniques, we get the best from both worlds. In this section, we give some examples of cases where the use of the hybrid visualization will drastically improve the result compared to any of the two individual techniques alone.

There are certain types of critical points, especially those with complex eigenvalues, that have a topology that is difficult to interpret in a visualization limited to using field lines. For instance, consider figure 5.28. This visualization has two critical points, both with complex eigenvalues, thus creating spiral structures. By visualizing the individual phase planes, as shown in figure 5.29, the Tangent LIC planes help reveal the spiral topology. Another example of complex critical point topology is shown in figure 5.30.

For a critical point with complex eigenvalues, a LIC plane centered in the critical point containing the real eigenvector will tell if the point has a saddle behaviour or not. Colored LIC planes can also help to separate the different spiraling critical points explained in section 4.9, which can not be separated with single-color field lines and grey scale LIC planes alone. For example, an attracting saddle focus and a repelling saddle focus would not be separable without the color coding. Points of these types are shown in figure 4.10. In some cases, for smooth vector fields and a good choice of seed points, it is possible to separate a focus with a saddle behaviour from one without, by studying the field lines alone. An example where this is difficult is given in figure 5.31. The seed points are well chosen, but the field around the critical point is too chaotic for a complete visual classification from the field lines alone. The color coded LIC plane shows that the phase plane containing the spiral has an inflow, and in the direction of the real eigenvector, which lies in the LIC plane, there is an outflow. This is one of the real strengths of Tangent LIC and field lines combined.

Consider a Tangent LIC visualization of a 3D node critical point. The color coding of the direction of flow only helps separate a repelling and an attracting 3D node. The ratio of the three eigenvalues of the system decides which direction the flow asymptotes to while leaving or approaching the critical point. The LIC planes alone will only depict the direction of flow tangential to the phase planes, which will be nodes in all three planes. Inside the area where the linearization is valid, any field line, except for the ones seeded on the lines defined by the eigenvector, asymptote toward the direction of the eigenvector corresponding to the eigenvalue of
Figure 5.28: Field line visualization of a complex topology with two critical points.

Figure 5.29: Using Tangent LIC planes to visualize the phase planes of the two critical points of figure 5.28.

5.4.1 Seeding Templates

In section 5.1, we presented a seeding strategy for field lines around critical points. This seeding strategy was primarily constructed for the phase planes of the critical point to help capture the topology in those planes. For the visualization we describe here it is important that the field lines give additional information not contained in the Tangent LIC planes. Therefore, one criteria is that the field lines should not be seeded directly in the phase planes. The seeding strategy we describe here is simplified as compared to the strategy in section 5.1. This is justified since the Tangent LIC planes already depict the topology in the phase planes.

For the hybrid visualization we also use a templated strategy. We divide the critical points into three different groups: 3D nodes, 3D saddles and critical points with complex eigenvalues. By a 3D node we mean a critical point whose phase-planes are of node topology in two or more planes, while a 3D saddle has saddle topology in two of the phase planes. The seeding templates
Figure 5.30: Left: A complex vector field topology displayed using Tangent LIC. Right: Adding field lines to the visualization reveals the surrounding vector field topology. There are two critical points in this visualization, one at the intersection of the three planes, and one inside the spiraling structure captured by the field lines.

Figure 5.31: An attracting saddle focus with and without a LIC plane. The LIC plane contains the real eigenvector and shows the directions of planar in- and outflow. The dataset is from [45].
Figure 5.32: A repelling node visualized using Tangent LIC, illuminated field lines and the hybrid approach.
for the three different groups of critical points are used in visualizations shown in figures 4.7-4.11 and described in the next three sections.

3D Saddle

For a repelling saddle we consider the eigenvector corresponding to the single negative eigenvalue. This eigenvector is the inflow direction of the critical point. On both sides of the critical point we seed on two circles around the line spanned by the eigenvector. This will divide the field lines evenly in the four quadrants (four on each side) defined by the two saddle planes sharing this eigenvector. The field lines will approach the plane containing the node and be deflected in the direction of the eigenvector corresponding to the largest eigenvalue. Field lines seeded in the different quadrants will leave the critical point from the same quadrants, but the distance between them will be larger than the distance between the seed points. This seeding template ensures that the field lines are spread evenly throughout the neighborhood of the critical point.

An attracting saddle is a repelling saddle with reversed flow directions as explained in section 4.9. Because of this, we seed in the exact same manner, but in this case in a circle around the single eigenvector corresponding to a positive eigenvalue. When integrating backwards, the field lines will be deflected the same way, and evenly spread in the plane of the attracting node.

3D Node

We use a spherical seeding template for 3D nodes. This template consists of seed points placed randomly on the surface of a sphere centered around the critical point. A sequence of candidate seeding points are generated. For each new candidate seed point, the Euclidean distance to the nearest of the previously seeded points on the sphere is calculated. If this distance is below a given threshold, the point is rejected, and a new candidate seed point is computed. This process is repeated until the sphere has a predetermined number of seed points.

The radius \( \rho \) of the sphere is determined as a fraction of the template scale \( D_{\text{scale}} \) defined in section 5.1.1. A uniform random number generator is used to generate two randomized values \( \phi \in [0, \pi] \) and \( \theta \in [0, 2\pi] \). The tuple \( (\rho, \phi, \theta) \) is the randomized location of a candidate seeding point on the surface of a sphere, expressed in spherical coordinates. This location is transformed into Cartesian coordinates using the following transformation:

\[
\begin{align*}
x &= \rho \sin(\phi) \cos(\theta) \\
y &= \rho \sin(\phi) \sin(\theta) \\
z &= \rho \cos(\phi).
\end{align*}
\] (5.20)

After the transformation into Cartesian coordinates, the Euclidean distance to the nearest, previously seeded point can be computed, and the process repeated until a predetermined number of seed points has been generated.

Complex Eigenvalues

The critical points having complex eigenvalues can be divided into two subgroups: focus and center. We use two different seeding templates for these two subgroups. For the center topology we seed along two lines parallel to either the real or imaginary vectors of the pair of complex eigenvectors, offset in the two normal directions of the phase plane. We limit the seeding so that points are only placed on one side of the critical point as shown in the left image of figure 5.3 on page 56. For the focus topology we seed along three lines parallel to either the real or imaginary vector of the complex conjugate pair of eigenvectors. One of the lines is located in
the phase plane, while the two other lines are offset in the two normal directions of the phase plane. The reason for seeding in the phase plane for this topology is to be able to hide the Tangent LIC plane in a visualization while still showing the characteristic flat, spiral topology of the field lines in the phase plane.
Chapter 6

Conclusion

Classification and visualization of critical points in 3D vector fields is a challenging task. In this thesis, we have developed a full set of methods ranging from locating critical points, to classification and visualization techniques.

To achieve a good visualization of critical points, it is important that the methods for extracting and classifying them are robust. For locating critical points, we improved upon the analytic method introduced by Gjøystdal [11]. The improved analytic method proved itself a viable tool for locating critical points, both in terms of effectiveness and robustness.

We have proposed a classification method for critical points based on phase planes. In a local region around a critical point, the phase planes inherit characteristic topologies that we have used in the classification of critical points. For a given critical point, these phase plane topologies are combined to yield a full 3D classification. The scheme also covers degenerate cases.

Building on the basis given by the classification scheme, we introduced a new set of seeding templates for field line visualization of critical points. The field lines are rendered using the illuminated field line technique [38], which we have implemented to run as a vertex shader program on a graphics processor.

We have also developed and implemented a method called Tangent LIC on the GPU for computing a LIC texture of the tangential component of a 3D vector field in a slicing plane. This visualization technique was used to visualize the phase planes of critical points. We achieved a visualization in which the LIC texture of the slicing plane is updated in real-time, allowing for interactive exploration of the visualization. This visualization method was combined with illuminated field lines in a hybrid visualization. The hybrid visualization managed to clearly capture both the details of the phase planes of the critical points, as well as the surrounding vector field topology.

Further Work

There are many possible extensions to the work presented here. The analytic method for locating critical points can be explored in terms of creating an analytic method for locating higher-order critical points. A parallel implementation on multiple CPUs of the analytic method will increase the performance of the method even further. It would also be interesting to see if the method can be extended to locate critical lines in vector fields, or track critical points in time-dependent vector fields.

The visualizations of critical points can be extended to multi-field visualizations, where multiple vector fields, or combinations of vector fields and scalar fields, are visualized together.
This would be useful, for instance, in studying the relationship between critical points and other qualitative features of a vector field.
Bibliography


