Closed Streamlines in Flow Visualization

Vom Fachbereich Informatik
der Universität Kaiserslautern
zur Erlangung des akademischen Grades

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

genehmigte Dissertation

von
Dipl.-Inform. Thomas Wischgoll

Kaiserslautern
25. Juni 2002

1. Berichterstatter: Prof. Dr. Hans Hagen
2. Berichterstatter: Prof. Dr. Eduard Gröller
Vorsitzender der Promotionskommission: Prof. Dr. Otto Mayer
Dekan: Prof. Dr. Jürgen Avenhaus

Datum der wissenschaftlichen Aussprache: 18. Oktober 2002

To my parents

Department of Computer Science, University of Kaiserslautern, Germany

> One thing I have learned in a long life: that all our science, measured against reality, is primitive and childlike and yet it is the most precious thing we have.
>
> *Albert Einstein (1879–1955)*

Acknowledgments

I want to thank all the people who helped me to accomplish this work during the last three and a half years. I thank my advisor, Prof. Dr. Hans Hagen, for giving me the opportunity to work on this interesting topic and letting me join international meetings and conferences in order to get in touch with other scientists. I also thank Dr. Gerik Scheuermann for providing me with some starting points and helping me throughout this work. Without him this work would not have been possible.

During my studies I was glad to have such helpful colleagues like Holger Burbach and Xavier Tricoche. I also thank all the members of the project FAnToM who provided the basis for my implementations. A special thank goes to Mady Gruys for creating a positive atmosphere and helping me in all non-scientific problems. Additionally, I want to thank Inga Scheler for proof reading this work and helping me to get it printed. Last but not least I would like to thank my parents who always supported me over the years.

Part of this work was funded by DFG (Deutsche Forschungsgemeinschaft) and Land Rheinland-Pfalz, Germany.

Abstract

Vector fields occur in many of the problems in science and engineering. In combustion processes, for instance, vector fields describe the flow of the gas. This process can be enhanced using vector field visualization techniques. Also, wind tunnel experiments can be analyzed. An example is the design of an air wing. The wing can be optimized to create a smoother flow around it. Vector field visualization methods help the engineer to detect critical features of the flow. Consequently, feature detection methods gained great importance during the last years.

Topological methods are often used to visualize vector fields because they clearly depict the structure of the vector field. In previous publications about topological methods closed streamlines are neglected. Since closed streamlines can behave in exactly the same way as sources and sinks they are an important feature that cannot be ignored anymore.

To accomplish this, this work concentrates on detecting this topological feature. We introduce a new algorithm that finds closed streamlines in vector fields that are given on a grid where the vectors are interpolated linearly. We identify regions that cannot be left by a streamline. According to the Poincaré-Bendixson theorem there is a closed streamline in such a region if it does not contain any critical point. Then we identify the exact location using the Poincaré map. In contrast to other algorithms, this method does not presume the existence of a closed streamline. Consequently, this algorithm is able to really detect closed streamlines inside the vector field. A parallel version of this algorithm is also described to reduce computational time. The implementation scales reciprocally proportional to the CPU speed of the used computers.

In order to get a better understanding of closed streamlines we sketch the whole evolution of a closed streamline in time dependent flows. This results in a tube shaped visualization representing the closed streamline over time. The emerging and vanishing of the closed streamline can be easily investigated to get more insight into this feature.

In combustion processes closed streamlines in a three dimensional flow are a hint for recirculation zones. These zones describe regions inside the flow where the gas stays quite long. This is necessary for the gas to completely burn. Therefore, we also show how to detect this important feature in three dimensional vector fields.

Kurzfassung

Vektorfelder treten im Zusammenhang mit sehr vielen wissenschaftlichen und ingenieurmäßigen Problemen auf. Bei Verbrennungsvorgängen beispielsweise beschreiben Vektorfelder den Verlauf des einströmenden Gases. Dieser Vorgang kann mit Hilfe von Techniken der Vektorfeldvisualisierung verbessert werden. Ebenso lassen sich Windkanalexperimente analysieren. Als Beispiel sei das Design einer Tragfläche genannt. Der Flügel kann optimiert werden, um eine bessere Umströmung zu erreichen. Methoden der Vektorfeldvisualisierung helfen dem Ingenieur, kritische Eigenschaften der Strömung zu erkennen. Dementsprechend erlangten Methoden, die Merkmale der Strömung aufzeigen, in den letzten Jahren immer größere Bedeutung.

Topologische Methoden werden häufig eingesetzt, um Vektorfelder zu visualisieren, da sie sehr deutlich die Struktur des Vektorfeldes aufzeigen. In früheren Veröffentlichungen über topologische Methoden wurden geschlossene Stromlinien bisher vernachlässigt. Da geschlossene Stromlinien sich jedoch genauso verhalten können wie Quellen und Senken, stellen sie ein wichtiges Merkmal dar, das nicht weiter ignoriert werden kann.

Um diesen Mangel zu beseitigen, befasst sich die Arbeit mit dem Auffinden dieser topologischen Eigenschaft. Es wird ein neuartiger Algorithmus vorgestellt, der in der Lage ist, geschlossene Stromlinien in Vektorfeldern, die auf einem Gitter definiert sind und linear interpoliert werden, zu finden. Dazu wird nach Bereichen gesucht, die von einer Stromlinie nicht mehr verlassen werden können. Gemäß dem Poincaré-Bendixson-Theorem befindet sich eine geschlossene Stromlinie in diesem Bereich, falls er keine kritischen Punkte enthält. Anschließend wird die genaue Position mit Hilfe der Poincaré-Abbildung bestimmt. Im Gegensatz zu anderen Algorithmen setzt diese Methode nicht die Existenz einer geschlossenen Stromlinie voraus. Daher ist das hier vorgestellte Verfahren in der Lage, geschlossene Stromlinien auch tatsächlich aufzufinden. Eine Parallelisierung dieses Algorithmus wird ebenfalls beschrieben, um die benötigte Rechenzeit zu reduzieren. Die Laufzeit der Implementation ist dabei umgekehrt proportional zur CPU-Geschwindigkeit der verwendeten Computer.

Um ein besseres Verständnis für geschlossene Stromlinien zu bekommen, wird der gesamte Lebenszyklus geschlossener Stromlinien in zeitabhängigen Vektorfeldern aufgezeigt. Dies resultiert in einer röhrenförmigen Darstellung, die geschlossene Stromlinien über die Zeit repräsentiert. Die Entstehung und das Verschwinden der geschlossenen Stromlinie kann einfach untersucht werden, um mehr Einblick in dieses Merkmal zu erhalten.

Bei Verbrennungsvorgängen sind geschlossene Stromlinien in dreidimensionalen Strömungen ein Indiz für Rezirkulationsbereiche. Diese Bereiche beschreiben Regionen innerhalb der Strömung, in denen sich das Gas relativ lange aufhält. Dies ist notwendig, damit das Gas vollständig verbrennen kann. Aus diesem Grund wird zudem aufgezeigt, wie dieses wichtige Merkmal in dreidimensionalen Vektorfeldern gefunden werden kann.

Department of Computer Science, University of Kaiserslautern, Germany

Contents

Chapter 1

Introduction

Many of the problems in natural science and engineering involve vector fields. Fluid flows, electric and magnetic fields are nearly everywhere, therefore measurements and simulations of vector fields are increasing dramatically. As with other data, analysis is much slower and still needs improvement. Mathematical methods together with visualization can provide help in this situation. In most cases, the scientist or engineer is interested in integral curves of the vector field like streamlines in fluid flows or magnetic field lines. The qualitative nature of these curves can be studied with topological methods developed originally for dynamical systems. Especially in the area of fluid mechanics, topological analysis and visualization have been used with success [GLL91], [HH91], [Ken98], [SHJK00].

In visualization, topological methods mostly are not able to precisely show the existence of closed streamlines. Only stopping criteria like elapsed time, number of integration steps or the length of the streamline are used to prevent the algorithm from running forever. But closed streamlines play an important role in topological methods because they can act in the same way as sources or sinks; they can attract or repel the flow. Therefore there is a strong need for an algorithm that is able to detect this important topological feature.

Figure 1.1 shows an example of a closed streamline. The streamline is started in the center of the figure. After a short while the integrated streamline ends up in a loop so that the streamline cycles around and around. Consequently, the computation normally would not terminate without detecting this situation or using an imprecise stopping criterion like the ones mentioned before. The disadvantage with these imprecise stopping criteria is that we cannot distinguish between a streamline that spirals very slowly and a streamline that runs into a closed streamline. Therefore, the streamline is stopped to early in some cases. If we were able to really detect whether we end up in a closed streamline or not we can compute the whole streamline. In this case we additionally accelerate the integration process because we do not cycle around the closed streamline anymore until the stopping criterion is fulfilled.

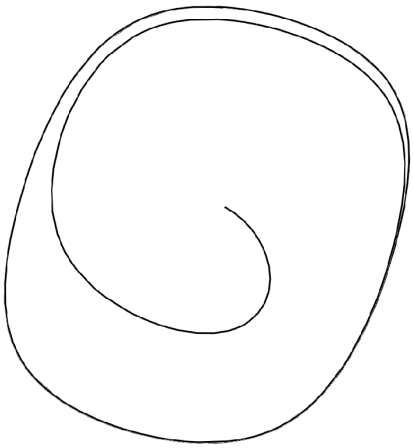


Figure 1.1: A closed streamline.

There are several applications for an algorithm that detects closed streamlines. In combustion processes, a special amount of time is necessary for the gas to burn completely. If we have a swirling jet where the gas is injected we have an inflow into a steady medium. Consequently, we get regions with high turbulence in the resulting flow. To locate areas where the gas stays very long we can use the closed streamlines in the flow indicating those areas which are called recirculation zones [Hai99].

To illustrate the situation we computed a visualization of a simulated inflow into a swirling jet. Figure 1.2 depicts the result. The flow is shown by a LIC image [CL93][SH95][HS98]. This method distorts a white noise image by smearing in the direction of the flow. The inflow can be clearly identified in the middle of the picture. The turbulent areas at the top and the bottom are emphasized also. Several closed streamlines can be found in these regions of the vector field drawn in white. Consequently, many recirculation zones can be identified by this method where chances are good that the gas is able to completely burn.

Another application is the search for the eye of hurricanes [WFL+00]. A hurricane consists of mainly two regions: the outer region where it has a great and destructive power with a circulating flow and the inner area where almost nothing happens. If we project the flow that describes the hurricane onto a horizontal plane these two regions are divided by a closed streamline. To locate this boundary we only have to compute the closed streamline of that vector field. As a result we have identified the eye of the hurricane.

After describing the theoretical background in the second chapter which is necessary for the understanding of the new algorithms that are described in this thesis, we explain the algorithm for detecting closed streamlines in a two dimensional vector field in the third chapter. This algorithm checks during the integration process if the streamline reaches an area that cannot be left. In such a case, we have proven according to the

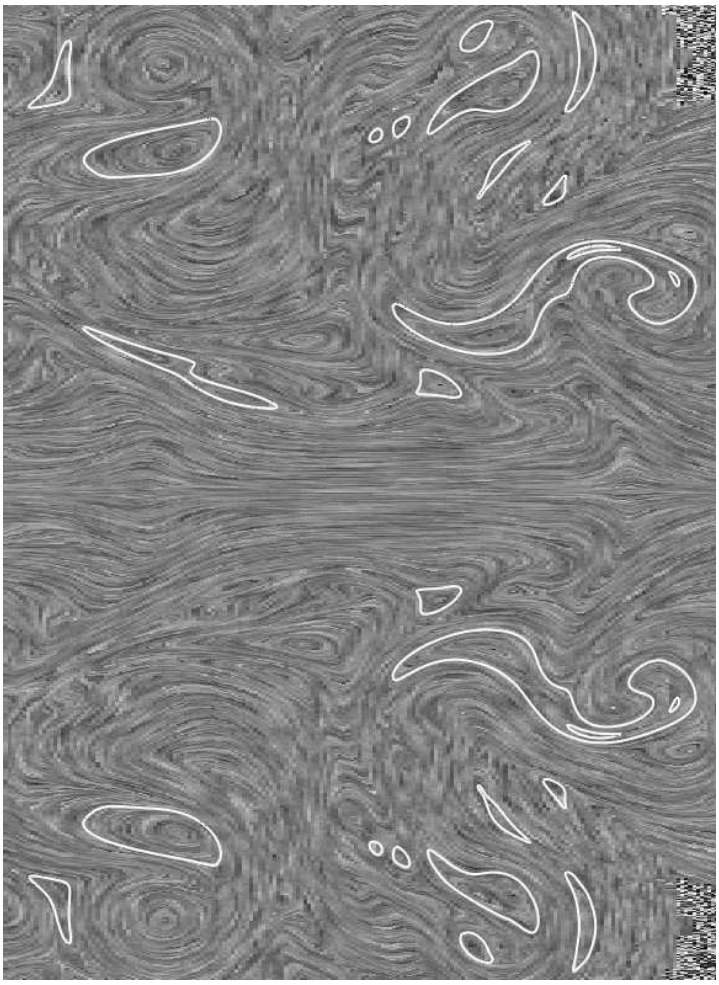


Figure 1.2: Recirculation zones in a swirling jet simulation.

Poincaré-Bendixson theorem that there exists a closed streamline in that area. To locate the exact position of this closed streamline the Poincaré map is used.

Since it is necessary to compute many streamlines to find every closed streamline in a given dataset, we show a parallelization of our algorithm that detects closed streamlines in chapter four. The streamlines that have to be computed are spread as different tasks to the various clients in a Linux cluster. If a closed streamline is detected the client sends back a visualization of that closed streamline to the server. The server displays all these closed streamlines. This facilitates a faster computation of all closed streamlines. The acceleration corresponds to the overall computation power of the whole Linux cluster.

Inspired by the books of Abraham and Shaw [AS82][AS83][AS84][AS88] we investigate the creation or vanishing of closed streamlines in chapter five. A vector field is interpolated over time so that closed streamlines can emerge in special situations.

These situations are called bifurcations. The closed streamlines are followed over time. This results in a tube shaped visualization that clearly shows the evolution of the whole closed streamline.

We also want to detect closed streamlines in a three dimensional vector field. Although the principle is quite similar to the two dimensional case there exist some essential differences. The test if a streamline is able to leave a region is far more difficult in the three dimensional case. This is described in detail in chapter seven.

Chapter 2

Theory of Vector Fields

This chapter introduces the fundamental theory which is needed for the following chapters. We mainly follow the description of Hirsch and Smale [HS74]. Other descriptions can be found in [Tri02][GH83][Guc00].

2.1 Fundamental Theory

In order to talk about vector fields we need a precise definition of what a vector field actual is.

Definition 2.1.1 (Vector field)
Let $W \subset \mathbb{R}^n$ be an open subset. An ***n-dimensional vector field*** *v is defined as a map*

$$v : W \to \mathbb{R}^n \quad .$$

As we can see from definition 2.1.1 a vector field gives us an n-dimensional vector at an arbitrary position inside W. Vector fields occur in many applications. For instance, we may have a flow in a wind tunnel experiment. This flow can be described by a *dynamical system*. If we have a massless particle located at a position x inside the flow, a dynamical system tells us where this particle is after a given time t. Therefore a dynamical system should be continuously differentiable or at least continuous and continuously differentiable in t.

Definition 2.1.2 (Dynamical system)
Let $W \subset \mathbb{R}^n$ be an open subset. A ***dynamical system*** *or* ***flow*** *is a C^1 map $\mathbb{R} \times W \xrightarrow{\phi} W$, where ϕ fulfills the following conditions:*

1. *$\phi(0)$ is the identity*
2. *$\phi(t) \circ \phi(s) = \phi(t+s)$ for all $t, s \in \mathbb{R}$*

We also write $\phi(t, x) = \phi_t(x)$.

There is a direct coherence between a dynamical system and a vector field shown by the next remark.

Remark 2.1.3
Let $W \subset \mathbb{R}^n$ *be an open subset and* ϕ *a dynamical system. Then there exists a vector field* $v : W \mapsto \mathbb{R}^n$ *that satisfies*

$$v(x) = \frac{d}{dt}\phi_t(x)\Big|_{t=0} \quad .$$

Thus, if $x_0 \in W$, $v(x_0)$ is the tangent vector to the curve defined by $t \to \phi_t(x_0)$ at $t = 0$. From another point of view we can start with a given vector field v. With a given point $x_0 \in W$, this leads us to the *Cauchy problem*.

Definition 2.1.4 (Cauchy problem)
Let v *be a vector field as in definition 2.1.1 and* $x_0 \in W$ *an arbitrary point. Then the* ***Cauchy problem*** *is defined by the differential equation*

$$\frac{d}{dt}x(t) = v(x(t))$$

with the so called ***initial condition*** $x(0) = x_0$.

Then the dynamical system ϕ satisfying the equation in remark 2.1.3 gives us the *solution curve* $x(t) = \phi_t(x)$ for the Cauchy problem.

Remark 2.1.5
The solution curve is also referred to as a ***streamline****, an* ***integral curve****, a* ***trajectory****, or an* ***orbit****.*

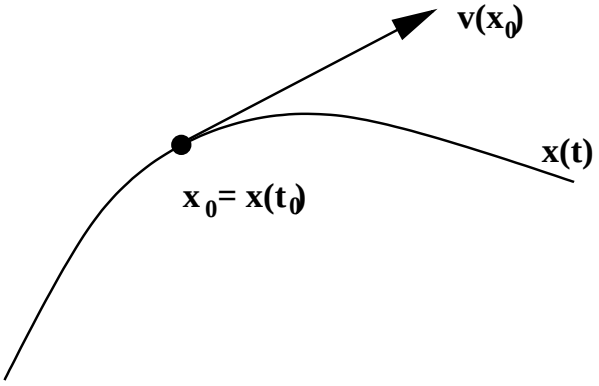


Figure 2.1: A solution curve is always tangential to the defining vector field.

From a geometrical point of view the trajectory $x(t)$ is a curve which is always tangential to the vector field v. This means that $\frac{d}{dt}x(t)$ equals $v(x(t))$ as shown in figure 2.1. For every point $x_0 \in W$ there exists a unique solution curve which is shown by the next theorem. But we need a short definition first.

Definition 2.1.6 (Lipschitz)
Let $W \subset \mathbb{R}^n$ be an open subset. Let further $v : W \to \mathbb{R}^n$ be a vector field as in definition 2.1.1. The vector field v is said to be ***Lipschitz*** *on W if there exists a constant K such that*

$$|v(x) - v(y)| \leq K|x - y|$$

for all $x, y \in W$. The constant K is called ***Lipschitz constant*** *for v.*

Theorem 2.1.7 (Existence and uniqueness)
Let v be a vector field as in definition 2.1.1 which is Lipschitz and $x_0 \in W$ an arbitrary point. Then there exists an $a > 0$ and a unique solution

$$x : (-a, a) \to W$$

of the Cauchy problem that satisfies the initial condition $x(0) = x_0$.

Proof:
See [HS74], pages 162 through 167. ❑

It follows directly from this theorem that streamlines cannot cross each other due to the uniqueness of solution curves.

Corollary 2.1.8 (Crossing streamlines)
Let v be a vector field as in definition 2.1.1 and $x_0, y_0 \in W$ two arbitrary points with $x_0 \neq y_0$. Let further $x(t)$ and $y(t)$ be the solution curves with initial conditions $x(0) = x_0$ respectively $y(0) = y_0$. Unless these two solution curves are not identical they do not intersect.

Proof:
Let x and y be two intersecting streamlines which are not identical. Let further $p \in W$ be the intersection point. Then there exist $t_1, t_2 \in \mathbb{R}$ with $x(t_1) = y(t_2) = p$. Then there are two streamlines starting at point p which contradicts to theorem 2.1.7. ❑

This corollary is of great importance since it shows that two different streamlines will never intersect. This feature will be exploited by our algorithm that detects closed streamlines.

Definition 2.1.9 (Phase portrait)
Let v be a vector field as in definition 2.1.1 and ϕ the dynamical system associated to v. Then the family of all solution curves represents the ***phase portrait*** *of the dynamical system ϕ.*

It is also possible to investigate vector fields over time. Therefore we define time dependent vector fields.

Definition 2.1.10 (Time dependent vector field)
Let $W \subset \mathbb{R}^n$ be an open subset. An ***n-dimensional time dependent vector field*** *v is defined as a map*

$$\begin{aligned} v : \mathbb{R} \times W &\to \mathbb{R}^n \\ (t, x) &\mapsto v(x) \end{aligned}$$

where t is the time parameter.

2.2 Data Structures

In most applications in Scientific Visualization the data is not given as a closed form solution. The same holds for vector fields. Usually, the vector fields result from a simulation or an experiment where the vectors are measured. In such a case, the vectors are given at only some points of the domain of the Euclidean space. These points are then connected by a grid. A special interpolation computes the vectors inside each cell of the grid. In this chapter we restrict ourselves to the few types of grids that we used in our algorithms in this section.

2.2.1 Triangular Grids

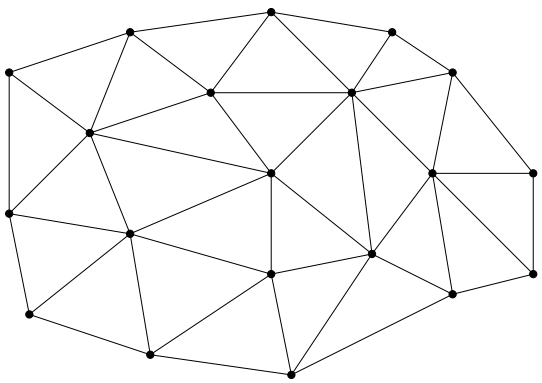


Figure 2.2: Triangular grid.

A very popular two dimensional grid type is the triangular grid. Figure 2.2 shows an example for such a grid. This grid type facilitates to connect an arbitrary point set. To get the vectors inside a cell we use an interpolation scheme based on barycentric

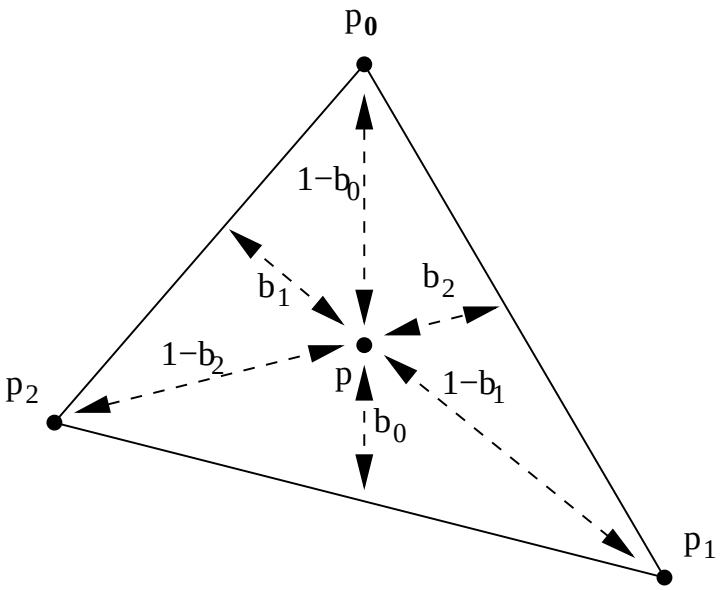


Figure 2.3: Barycentric coordinates.

coordinates. Figure 2.3 explains the configuration. Let p be the point where we want to interpolate the vector and p_0, p_1, and p_2 are the vertices of the triangle.

The barycentric coordinates b_0, b_1, and b_2 describe the distances between the point p_i and the edge which is opposite to the vertex with the same index. For the barycentric coordinates the equation $\sum_{i=0}^{2} b_i = 1$ holds. The point p can be expressed in the following way:

$$p = \sum_{i=0}^{2} b_i \cdot p_i \quad .$$

Let $v(p_i)$ be the vectors at the vertices of the triangle. Then we can interpolate the vector $v(p)$ at the point p in the same way:

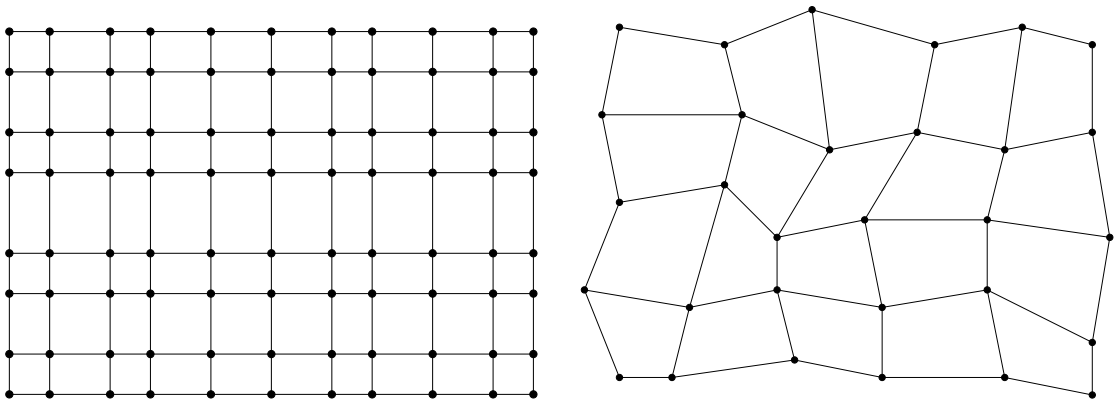
$$v(p) = \sum_{i=0}^{2} b_i \cdot v(p_i) \quad .$$

To compute the zeros inside the triangle we need to solve the following linear equation:

$$\sum_{i=0}^{2} b_i \cdot v(p_i) = 0 \quad .$$

Unless this system is degenerated, the solution is unique. Consequently, we get at most one zero depending on whether the solution point lies inside the triangle or not.

2.2.2 Quadrilateral Grids



(a) Rectilinear grid. (b) Curvilinear grid.

Figure 2.4: Quadrilateral grids.

There exist two different types of quadrilateral grids. The first one is the rectilinear grid. There every cell is a rectangle. The edges of the cells are orthogonal as shown in figure 2.4a. The other type is the curvilinear grid. Here the boundary between the cells is a curve consisting of points connected by straight lines. The boundary of more than one cell does not need to be a straight line anymore as can be seen in figure 2.4b. If we interpolate in such a cell we need to map it to a rectangular cell. This map ϕ is not linear. Often one speaks of mapping from physical space into computational space. Usually, a numerical method is used to do this mapping. Consequently, we can restrict ourselves to the rectangular case when explaining interpolation in this case.

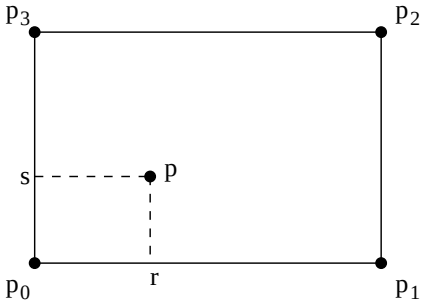


Figure 2.5: Local coordinates inside a rectangle.

We interpolate bilinearly inside each cell. Therefore, we introduce local coordinates (r, s) with $0 \leq s, r \leq 1$ inside the cell of the point p where we want to interpolate.

Figure 2.5 explains how this works. The vertex p_0 has local coordinates $(0,0)$, the vertex p_1 the coordinates $(1,0)$, the vertex p_2 corresponds to $(1,1)$, while the last vertex p_3 is located at $(0,1)$. Then we can use the following formula for the interpolation:

$$v(p) = (1-r)(1-s) \cdot v(p_0) + r(1-s) \cdot v(p_1) + rs \cdot v(p_2) + (1-r)s \cdot v(p_3) \quad .$$

2.2.3 Tetrahedral Grids

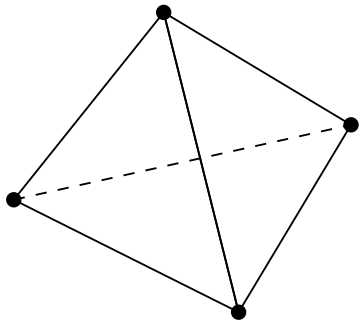


Figure 2.6: Tetrahedron.

A tetrahedral grid consists of several tetrahedrons as shown in figure 2.6. Consequently, a tetrahedral grid is a three dimensional grid. As with triangular grids an arbitrary point set can be connected using this grid type. The interpolation scheme works in an analogue way as the triangular case using barycentric coordinates. Consequently, there is at most one zero inside each tetrahedron, also, if the interpolating vector field is non-degenerate inside that tetrahedron.

2.2.4 Time dependent Data with Prism Cells

When dealing with time-dependent two-dimensional flows we can use the third dimension to represent time. We assume the vector field is given at time slices on a triangular grid. These time slices $v_i : W \to \mathbb{R}^2$ are connected using prism cells as shown in figure 2.7. To interpolate the vectors we consider the following map

$$\begin{array}{rcl} f : \mathbb{R} \times W & \longrightarrow & \mathbb{R} \times \mathbb{R}^2 \\ (t,x) & \mapsto & v(t,x) \end{array}$$

where W is the domain represented by the two dimensional grid of the time slices. Since we need consistency with the piecewise affine linear interpolation that would be applied on a 2D triangulation, we have to ensure that the restriction of the 3D interpolant to each time plane is piecewise affine linear, too. That means that, fixing the time

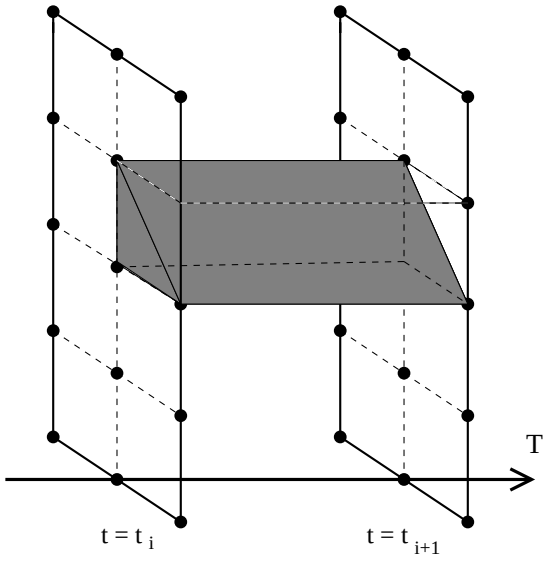


Figure 2.7: Time prism cell.

coordinate and taking it as a parameter, the interpolant must be affine linear. This is the reason why we choose the following interpolant inside each prism cell.

For a given prism cell lying between t_i and t_{i+1}, let $v_j(x) = A_j x + b_j$, $j \in \{i, i+1\}$ be the linear interpolation corresponding to the prism triangle faces lying in the planes $\{t = t_i\}$ and $\{t = t_{i+1}\}$ respectively. Then we define the interpolant over the whole prism cell by linear interpolation over time:

$$v(t,x) = \frac{t_{i+1} - t}{t_{i+1} - t_i} v_i(x) + \frac{t - t_i}{t_{i+1} - t_i} v_{i+1}(x)$$

where $t \in [t_i, t_{i+1}]$. This formula obviously ensures, for each fixed value of t, that $v(x,t)$ is affine linear in x.

2.3 Critical Points

Critical points are from a topological point of view an important part of vector fields. This special feature is described in more detail in this section. We first explain the general case and then study the linear case.

2.3.1 General Case

We start with the definition of critical points in the general case. Then we classify different types of singularities and talk about stability which is necessary to achieve a

meaningful physical interpretation of vector fields.

Definition 2.3.1 (Critical point)
Let $v : W \to \mathbb{R}^n$ be a vector field as in definition 2.1.1 which is continuously differentiable. Let further $x_0 \in W$ be a point where $v(x) = 0$. Then x_0 is called a ***critical point*** *of the vector field.*

Remark 2.3.2
There are several different terms for critical points. They are also known as ***singularities***, ***singular points***, ***zeros***, *or* ***equilibriums***.

2.3.1.1 Classification

Critical points can be classified using the eigenvalues of the derivation of the vector field. For instance, we can identify *sinks* that purely attract the flow in the vicinity while *sources* repel it purely.

Definition 2.3.3 (Sink)
Let v be a vector field as in definition 2.1.1 which is continuously differentiable and x_0 a critical point of v. Let further $Dv(x_0)$ be the derivation of the vector field v at x_0. If all eigenvalues of $Dv(x_0)$ have negative real parts, x_0 is called a ***sink***.

The following theorem shows that sinks really have an attracting property.

Theorem 2.3.4
Let $v : W \to \mathbb{R}^n$ be a vector field and x_0 a sink. Let further ϕ be the corresponding dynamical system. Let us assume the real part of every eigenvalue is less than $-c$, $c > 0$. Then there exists a neighborhood $U \subset W$ of x_0 such that

1. *$\phi_t(x) \in U$ for all $x \in U$, $t > 0$.*
2. *There is an Euclidean norm on $\mathbb{R}^n$ such that*

$$|\phi_t(x) - x_0| \leq e^{-tc}|x - x_0|$$

   for all $x \in U$, $t \geq 0$.

3. *For any norm on $\mathbb{R}^n$, there is a constant $B > 0$ such that*

$$|\phi_t(x) - x_0| \leq Be^{-tc}|x - x_0|$$

   for all $x \in U$, $t \geq 0$.

Proof:
See [HS74], pages 181 and 182. ❑

Corollary 2.3.5
Let v, ϕ, and x_o be as in the previous theorem. Then there exists a neighborhood $U \subset W$ of x_0 so that $\phi_t(x)$ converges to x_0:

$$\phi_t(x) \to x_0 \text{ as } t \to \infty \text{ for all } x \in U$$

In the same way we can define sources.

Definition 2.3.6 (Source)
Let v be a vector field as in definition 2.1.1 which is continuously differentiable and x_0 a critical point of v. Let further $Dv(x_0)$ be the derivation of the vector field v at x_0. If all eigenvalues of $Dv(x_0)$ have positive real parts, x_0 is called a ***source****.*

2.3.1.2 Stability

Since in computer science absolute exact calculation is not possible due to numerical errors we need some sort of stability if we really want to classify critical points algorithmically. A critical point that changes its behavior even when the vector field is slightly perturbed does not have a very significant meaning in a physical sense.

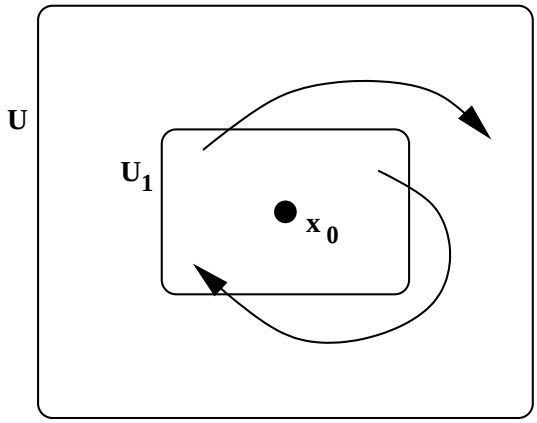


Figure 2.8: A critical point that is stable.

Definition 2.3.7 (Stable critical point)
Let $v : W \to \mathbb{R}^n$ be a vector field as in definition 2.1.1 which is continuously differentiable and x_0 a stable critical point of v. If for every neighborhood $U \subset W$ of x_0 there is a neighborhood $U_1 \subset U$ of x_0 such that every streamline $x(t)$ with $x(0) \in U_1$ is defined and $x(t) \in U$ for all $t > 0$ then x_0 is called a ***stable critical point****.*

Figure 2.8 illustrates a stable configuration.

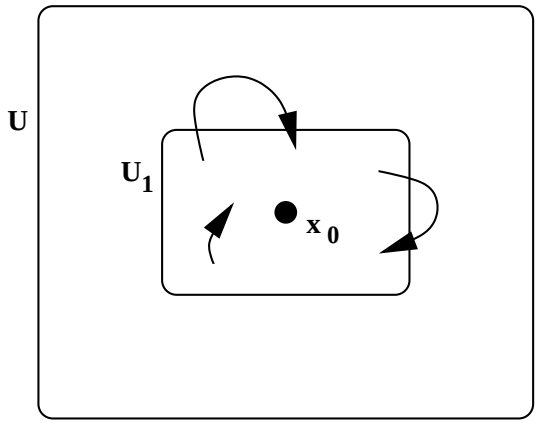


Figure 2.9: An asymptotically stable critical point.

Definition 2.3.8 (Asymptotically stable critical point)
Let v, U, and U_1 be as in the previous definition. If in addition U_1 can be chosen so that $\lim_{t\to\infty} x(t) = x_0$ then x_0 is called an ***asymptotically stable critical point****.*

In figure 2.9 we sketch this situation.

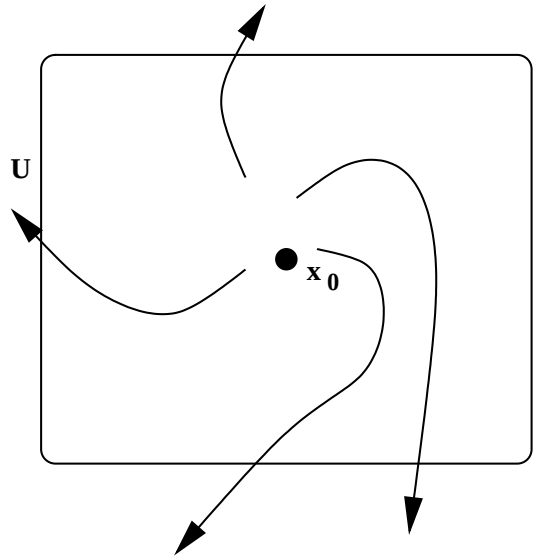


Figure 2.10: A critical point that is unstable.

Definition 2.3.9 (Unstable critical point)
Let $v : W \to \mathbb{R}^n$ be a vector field as in definition 2.1.1 which is continuously differentiable and x_0 a stable critical point of v. We call a critical point that is not stable an ***unstable critical point****. This means that there is a neighborhood $U \subset W$ of x_0 such that for every*

neighborhood $U_1 \subset U$ of x_0 there is at least one streamline $x(t)$ starting at $x(0) \in U_1$ which does not completely lie in U.

Figure 2.10 shows an unstable critical point.

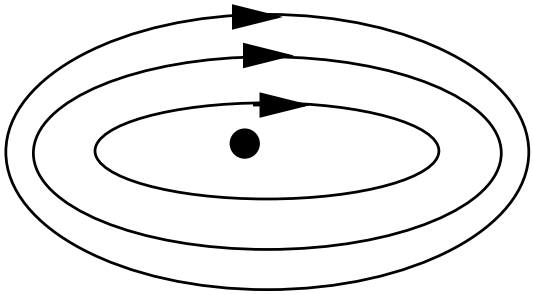


Figure 2.11: A critical point that is stable but not asymptotically stable.

For example, a sink is an asymptotically stable critical point and therefore stable. An example of a critical point that is stable but not asymptotically stable is shown in figure 2.11. All streamlines surround the critical point elliptically. This configuration is rather critical because the slightest perturbation will change the critical point into a source or a sink. Therefore, we want to distinguish between such numerically critical situations and numerically stable ones.

Theorem 2.3.10
Let v be a vector field as in definition 2.1.1 which is continuously differentiable and x_0 a stable critical point of v. Then no eigenvalue of $Dv(x_0)$ has positive real part.

Proof:
See [HS74], pages 187 and 188. ❑

To have a common term for such numerically stable configurations we use the notion of *hyperbolicity*.

Definition 2.3.11 (Hyperbolic critical point)
Let v be a vector field as in definition 2.1.1 which is continuously differentiable and x_0 a critical point of v. If the derivative $Dv(x_0)$ has no eigenvalue with real part zero the critical point is called ***hyperbolic****.*

Corollary 2.3.12
A hyperbolic critical point is either unstable or asymptotically stable.

This corollary shows that hyperbolic critical points avoid numerically critical situations. These points can be detected algorithmically since the behavior does not significantly change if there is a numerical error that is small enough.

2.3.2 Linear Case

Since we use linear or bilinear interpolation in our algorithms we want to analyze the linear case in more detail. This also gives a better insight into the different types of critical points. Therefore we examine critical points in linear vector fields. First of all, we need to define what we exactly mean by a linear vector field.

Definition 2.3.13 (Linear vector field)
Let $W \subset \mathbb{R}^n$ be an open subset. A vector field

$$v : W \to \mathbb{R}^n$$

is called linear, if there exists a linear map

$$A : W \to \mathbb{R}^n$$

and a vector $b \in \mathbb{R}^n$ such that

$$v(x) = Ax + b \quad \forall x \in W$$

if in addition $b = 0$ then v is called ***homogeneous linear****.*

To get a better insight into linear vector fields we investigate the phase portrait of the different types of linear vector fields. If we restrict ourselves to the hyperbolic case where $\det A \neq 0$ the vector b only gives a displacement so that we can neglect it in our consideration. Nielson [NJ99] summarized all different cases that are possible. A linear vector field can have at most one critical point due to the linearity. In order to get the phase portrait we have to solve the Cauchy problem

$$\tfrac{d}{dt}x(t) = Ax$$

with initial condition $x(0) = k$, $k \in \mathbb{R}^n$.

Lemma 2.3.14
Let v be a homogeneous linear vector field, which is described by the matrix $A \in Mat(n \times n)$. Then there exists a solution for the differential equation

$$\tfrac{d}{dt}x(t) = Ax(t) \text{ with initial condition } x(0) = k \in \mathbb{R}^n \tag{2.1}$$

which is given by:

$$x = e^{tA}k \text{ with } e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!} \tag{2.2}$$

Proof:
Compute the derivation $\frac{d}{dt}x(t)$:

$$\frac{d}{dt}e^{tA}k = k \cdot \frac{d}{dt}e^{tA} = k \cdot Ae^{tA}$$

since the derivation of e^{tA} can be computed as follows:

$$\begin{aligned}\frac{d}{dt}e^{tA} &= \lim_{h\to 0}\frac{e^{(t+h)A} - e^{tA}}{h} \\ &= lim_{h\to 0}\frac{e^{tA}e^{hA} - e^{tA}}{h} \\ &= e^{tA}\lim_{h\to 0}\frac{e^{hA} - I}{h} \\ &= e^{tA} \cdot A\end{aligned}$$

❑

Let us have a closer look at two dimensional linear vector fields which can be described by a matrix $A \in Mat(2 \times 2)$. Then, we can distinguish between different cases where we are able to compute the derivation.

Lemma 2.3.15
Let $A \in Mat(2 \times 2)$ be a two dimensional matrix. Then there is an invertible matrix P such that $B = PAP^{-1}$, where B corresponds to one of the following three different types. $\lambda, \mu \in \mathbb{C}$ are the eigenvalues of A.

Type one: *A is diagonalizable:*

$$B = \begin{pmatrix}\lambda & 0 \\ 0 & \mu\end{pmatrix}$$

Type two: *λ and μ have non zero imaginary part:*

$$B = \begin{pmatrix}a & -b \\ b & a\end{pmatrix}$$

Type three: *$\lambda = \mu$ and A is not diagonalizable:*

$$B = \begin{pmatrix}\lambda & 0 \\ 1 & \lambda\end{pmatrix}$$

The next subsections describe the different types in detail. The differential equation is solved to sketch the phase portrait. We assume that the matrices of the vector fields are given in the form as shown in lemma 2.3.15.

2.3.2.1 Linear Vector Fields of Type One

Lemma 2.3.16

Let v be a vector field of type one as described in lemma 2.3.15. Then the following equation holds for the streamline x, where $k = \binom{k_1}{k_2}$ is a point on the streamline.

$$x(t) = \begin{pmatrix} e^{t\lambda}k_1 \\ e^{t\mu}k_2 \end{pmatrix} \tag{2.3}$$

Proof:

A phase portrait of a vector field of type one can be described in principle in the following way.

$$A = \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix}$$

Compute e^{tA}:

$$\begin{aligned} e^{tA} &= \sum_{k=0}^{\infty} \frac{(tA)^k}{k!} \\ &= \sum_{k=0}^{\infty} \frac{\begin{pmatrix} t\lambda & 0 \\ 0 & t\mu \end{pmatrix}^k}{k!} \\ &= \sum_{k=0}^{\infty} \begin{pmatrix} \frac{t\lambda}{k!} & 0 \\ 0 & \frac{t\mu}{k!} \end{pmatrix}^k \\ &= \begin{pmatrix} \sum_{k=0}^{\infty} \frac{(t\lambda)^k}{k!} & 0 \\ 0 & \sum_{k=0}^{\infty} \frac{(t\mu)^k}{k!} \end{pmatrix} \\ &= \begin{pmatrix} e^{t\lambda} & 0 \\ 0 & e^{t\mu} \end{pmatrix} \end{aligned}$$

Altogether we get the following equation describing the streamline.

$$x(t) = \begin{pmatrix} e^{t\lambda}k_1 \\ e^{t\mu}k_2 \end{pmatrix}$$

❑

Therefore, we mainly get three different cases besides changing the orientation of the vector field.

1st case: ($\lambda > 0 > \mu$) In this case we get the so called **saddle singularity**, which is shown in figure 2.12.

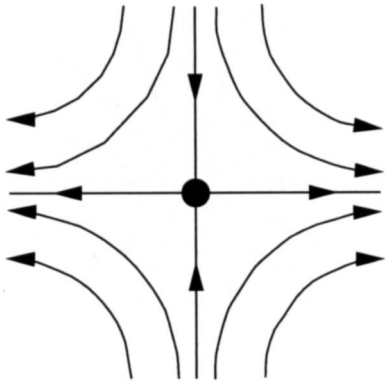


Figure 2.12: A saddle singularity.

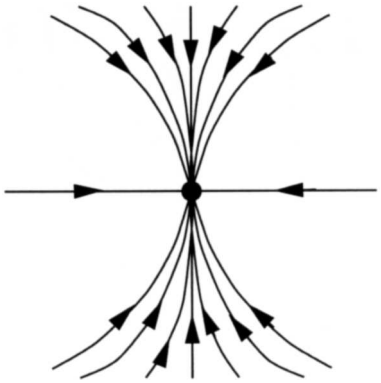


Figure 2.13: A node singularity.

2nd case: ($\lambda < \mu < 0$) An example for this case is the so called **node singularity** shown in figure 2.13.

3rd case: ($\lambda = \mu < 0$) Figure 2.14 shows such a **focus singularity**.

2.3.2.2 Linear Vector Fields of Type Two

The matrix A that represents the vector field mathematically describes a rotation and a scaling. This can be shown easily when we define a rotational angle $\Theta := \arccos(\frac{a}{r})$ where we set $r := \sqrt{a^2 + b^2}$. We can deduce that:

$$a = r \cos \Theta \tag{2.4}$$

$$b = r \sin \Theta \tag{2.5}$$

Then we can write the matrix A as follows:

$$A = \begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix} \cdot \begin{pmatrix} \cos \Theta & -\sin \Theta \\ \sin \Theta & \cos \Theta \end{pmatrix} \tag{2.6}$$

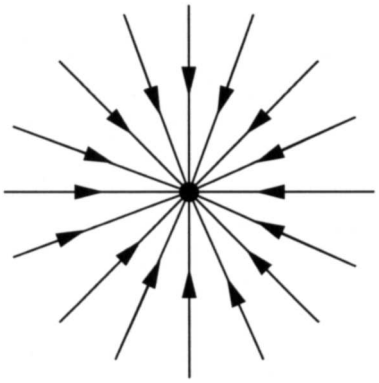


Figure 2.14: A focus singularity.

Lemma 2.3.17
Let v be a vector field of type two. Then the following equation describes a streamline where $k = \binom{k_1}{k_2}$ is a point on the streamline.

$$x(t) = e^{ta} \cdot \begin{pmatrix} k_1 cos(tb) - k_2 sin(tb) \\ k_1 sin(tb) + k_2 cos(tb) \end{pmatrix} \tag{2.7}$$

Proof:
We interpret the map T given by the matrix A algebraically by identifying $\mathbb{R}^2$ with the complex space $\mathbb{C}$.

$$(x, y) \leftrightarrow x + iy \tag{2.8}$$

We get the following correspondence for T:

$$\begin{array}{ccc} (x,y) & \longleftrightarrow & x+iy \\ \updownarrow T & & \updownarrow \text{Multiplying with } a+ib \\ (ax-by, bx+ay) & \longleftrightarrow & (ax-by)+i(bx+ay) \end{array} \tag{2.9}$$

In the same way there is a correspondence $e^A \leftrightarrow e^{a+b}$. This results with $e^A = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix}$ in the following scheme:

$$\begin{array}{ccc} (x,y) & \longleftrightarrow & x+iy \\ \updownarrow e^A & & \updownarrow e^{a+ib} \\ (a_1x+a_2y, a_3x+a_4y) & \longleftrightarrow & e^a(x\cos b - y\sin b + i(x\sin b + y\cos b)) \end{array} \tag{2.10}$$

By comparing the coefficients we can conclude that the matrix e^A can be represented as follows:

$$e^A = e^a \cdot \begin{pmatrix} \cos b & -\sin b \\ \sin b & \cos b \end{pmatrix}$$

According to lemma 2.3.14 the following equation holds for a streamline containing the point $k = \binom{k_1}{k_2}$.

$$x(t) = e^{ta} \cdot \begin{pmatrix} k_1 cos(tb) - k_2 sin(tb) \\ k_1 sin(tb) + k_2 cos(tb) \end{pmatrix} \tag{2.11}$$

❑

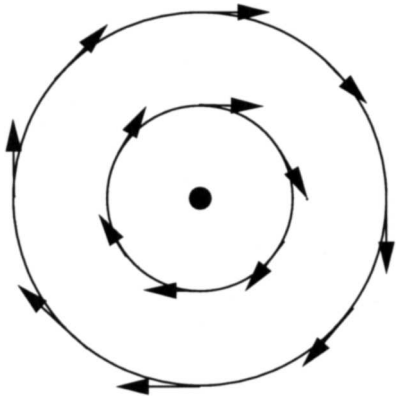


Figure 2.15: A center singularity.

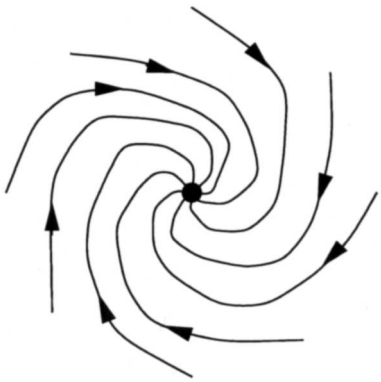


Figure 2.16: A spiral singularity.

With this equation we can see how the streamlines behave in such a vector field. If we have $a = 0$, the vector field describes simple circles as shown in figure 2.15, while we get a spiral shaped phase portrait if we set $a \neq 0$ as sketched in figure 2.16.

2.3.2.3 Linear Vector Fields of Type Three

Lemma 2.3.18

Let v be a vector field of type three and A the corresponding matrix where $v(x) = Ax$ and $A = \begin{pmatrix} \lambda & 0 \\ 1 & \lambda \end{pmatrix}$. Then we can describe a streamline containing the point $k = \binom{k_1}{k_2}$ with the following equation:

$$x(t) = e^{t\lambda} \cdot \begin{pmatrix} k_1 \\ k_1 t + k_2 \end{pmatrix} \tag{2.12}$$

Proof:

The matrix A can be split up in the following way:

$$A = \begin{pmatrix} \lambda & 0 \\ 1 & \lambda \end{pmatrix} = \lambda \cdot I + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

For the matrix $M = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$ the following equation holds which can be easily computed.

$$M^2 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = 0$$

Consequently, we get $M^k = 0$ for all $k \geq 2$.

Then we can compute e^{tA} as follows:

$$\begin{aligned} e^{tA} &= e^{t(\lambda I + M)} \\ &= e^{t\lambda I + \begin{pmatrix} 0 & 0 \\ t & 0 \end{pmatrix}} \\ &= e^{t\lambda I} \cdot e^{\begin{pmatrix} 0 & 0 \\ t & 0 \end{pmatrix}} \\ &= e^{t\lambda I} \cdot (I + \begin{pmatrix} 0 & 0 \\ t & 0 \end{pmatrix}) \text{ , using the above equation} \\ &= e^{t\lambda} \cdot \begin{pmatrix} 1 & 0 \\ t & 1 \end{pmatrix} \end{aligned}$$

Therefore, the following equation describes a streamline containing the point $k = \binom{k_1}{k_2}$.

$$x(t) = e^{t\lambda} \cdot \begin{pmatrix} k_1 \\ k_1 t + k_2 \end{pmatrix} \tag{2.13}$$

❑

Figure 2.17 shows an example for such an *improper node singularity.*

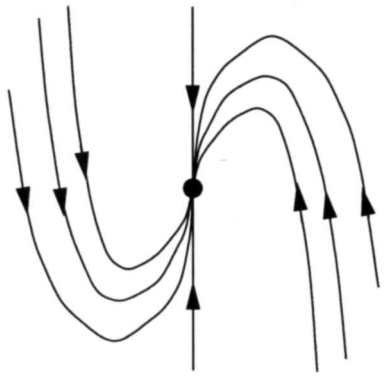


Figure 2.17: An improper node singularity.

2.4 Streamline Computation

In this section we describe the computation of streamlines. Since the vector field is given on a triangular, quadrilateral, or tetrahedral grid the vectors inside the cells are interpolated linearly respectively bilinearly. In order to compute a streamline we have to solve the Cauchy problem where the initial condition is given by the starting point of the streamline. Therefore we need to solve a differential equation. Consequently, the streamlines itself have to be calculated using ODE solvers like for instance Runge-Kutta. The streamlines can be integrated in positive or negative direction starting at the given starting point. To integrate in negative direction we only need to invert the vector field. In addition, it is possible to compute streamlines exactly inside triangular cells.

2.4.1 Numerical Computation

For numerical integration we use standard methods that can be found in the numerical literature [Tri02][Feh69][PTVF92][Guc00]. We favor predictor-corrector methods like Runge-Kutta method with adaptive stepsize. An optimized implementation for a fifth order Runge-Kutta method with adaptive stepsize can be found in [PTVF92]. These methods only use the interpolation method inside the cells but do not depend on a special type of grid.

2.4.2 Exact Computation

On a triangular grid the vector field is interpolated linearly. Inside a triangular cell we can represent the vector field as a single linear vector field as in subsection 2.3.2. In this subsection we also explained an exact solution of the differential equation that has to be solved in order to compute a streamline. This method was first introduced by Nielson [NHM97]. Consequently, we can calculate a streamline starting at an arbitrary

starting point using these formulas inside a triangle. When the streamline leaves the triangle we determine the intersection with one of the edges of the triangle numerically. Then we can start the integration process in the neighboring cell at that intersection. This way we can go from one triangle to another.

2.5 Closed Streamlines

When computing streamlines it often happens that the streamline computation does not terminate. This is mostly due to closed streamlines where the streamline ends up in a loop that cannot be left. These closed streamlines are introduced and explained in this section. More about the theoretical background can be found in several books [YqSlLs+86][Rou98].

2.5.1 Limit Sets

The topological analysis of vector fields considers the asymptotic behavior of streamlines. There we have two different kind of so called limit sets, the origin set or α-limit set of a streamline and the end set or ω-limit set.

Definition 2.5.1 (α-limit set)
Let s be a streamline in a given vector field v. Then we define the ***α-limit*** *set as the following set:*

$$\{p \in \mathbb{R}^2 \,|\exists (t_n)_{n=0}^{\infty} \subset \mathbb{R}, t_n \to -\infty, \lim_{n\to\infty} s(t_n) \to p\}$$

Definition 2.5.2 (ω-limit set)
Let s be a streamline in a given vector field v. Then we define the ***ω-limit*** *set as the following set:*

$$\{p \in \mathbb{R}^2 \,|\exists (t_n)_{n=0}^{\infty} \subset \mathbb{R}, t_n \to \infty, \lim_{n\to\infty} s(t_n) \to p\}$$

Remark 2.5.3
Let v be a vector field as in definition 2.1.1. We speak of an α- or ω-limit set L of v if there exists a streamline s in the vector field v that has L as α- or ω-limit set.

If the α- or ω-limit set of a streamline consists of only one point, this point is a critical point. The most common case of a α- or ω-limit set in a planar vector field containing more than one inner point of the domain is a closed streamline which is declared in the next definition. Figure 2.18 shows an example for α- and ω-limit sets. Here we have a critical point and a closed streamline. The critical point and the closed streamline are their own α- and ω-limit set. For every other streamline the closed streamline is the ω-limit set. If the streamline starts inside the closed streamline the critical point is the

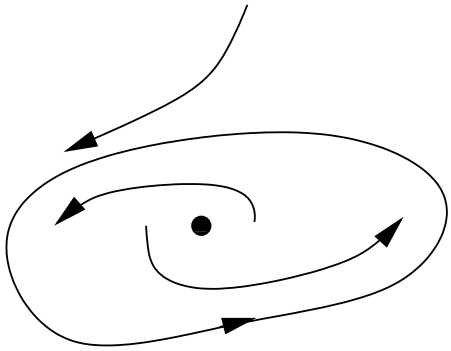


Figure 2.18: Example for α- and ω-limit sets.

α-limit set. Otherwise the α-limit set is empty. Now that we showed an example for a closed streamline let us give a precise definition.

Definition 2.5.4 (Closed streamline)
Let v be a vector field as in definition 2.1.1. A ***closed streamline*** $\gamma : \mathbb{R} \to \mathbb{R}^n$, $t \mapsto \gamma(t)$ *is a streamline of a vector field v such that there is a $t_0 \in \mathbb{R}$ with*

$$\gamma(t + nt_0) = \gamma(t) \quad \forall n \in \mathbb{N}$$

and γ not constant.

Remark 2.5.5
There are several different terms describing a closed streamline. The terms ***limit cycle***, ***closed orbit***, *and closed streamline are equivalent.*

Similar to critical points we define *asymptotically stability* of closed streamlines. If a closed streamline is asymptotically stable it is attracting.

Definition 2.5.6 (Asymptotically stability of closed streamlines)
Let $v : W \to \mathbb{R}^n$ be a vector field as in definition 2.1.1 that is continuously differentiable. Let further ϕ be the corresponding dynamical system and $\gamma \subset W$ a closed streamline. If for every neighborhood $U \subset W$ with $\gamma \subset U$ there is a neighborhood $U_1 \subset U$ with $\gamma \subset U_1$ such that $\phi_t(x) \in U$ for all $x \in U_1$ and $t > 0$ and

$$\lim_{t \to \infty} \min\{\|\phi_t(x) - z\| | z \in \gamma\} = 0$$

then γ is called ***asymptotically stable*** *closed streamline.*

This means that an asymptotically stable closed streamline attracts the flow inside a special neighborhood. It also follows from this definition that an asymptotically stable closed streamline is isolated from other closed orbits. In the same way there are closed streamlines that are repelling. For instance, by inverting the vector field we can turn an attracting closed streamline into a repelling one.

2.5.2 Poincaré Map

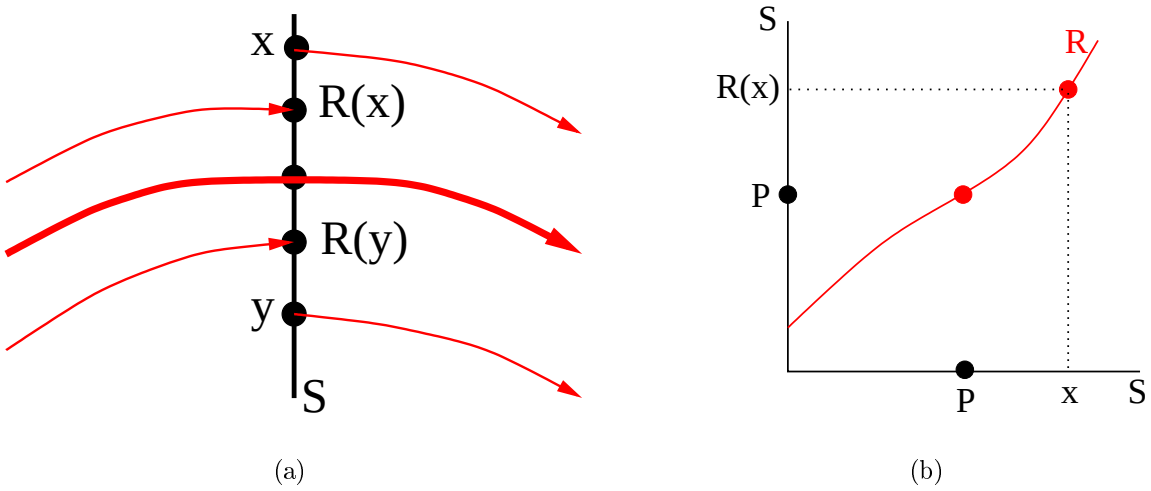


Figure 2.19: Poincaré section (a) and Poincaré map (b).

Let us assume we have a two dimensional vector field containing one limit cycle. Then we can choose a point P on the limit cycle and draw a *cross section* S which is a line segment not parallel to the limit cycle across the vector field. This line is called a *Poincaré section*. If we start a streamline at an arbitrary point x on S and follow it until we cross the Poincaré section S again, we get another point $R(x)$ on S. This results in the *Poincaré map* R. Figure 2.19 illustrates the situation. The left part shows the Poincaré section with the limit cycle in the middle drawn with a thicker line, while the right part displays the Poincaré map itself. Obviously the point P on the limit cycle is mapped onto itself. Consequently, it is a fixed point of the Poincaré map.

Let us precise this in some definitions:

Definition 2.5.7 (Cross section)
Let v be a vector field as in definition 2.1.1 and $S \subset \mathbb{R}^n$ an open set on a hyperplane of dimension $n-1$ that is transverse to v. Transverse to v means that $v(x) \notin S$ for all $x \in S$. Then S is called a ***cross section****.*

Definition 2.5.8 (Poincarè map)
Let v be a vector field and ϕ the dynamical system belonging to v. Let further be S a cross section that intersects a closed streamline at a point P. Then the ***Poincarè map*** *is defined as the map $R : S \to S$ such that*

$$x \mapsto \phi_t(x) \quad ,$$

where t is the time the streamline started at x needs to intersect the cross section again after one turn.

Remark 2.5.9
It is obvious that the point P on the closed streamline is a fixed point of the Poincarè map.

2.5.3 The Poincaré-Bendixson Theorem

In this subsection we show a fundamental result which makes it easier to find closed streamlines in a two dimensional vector field. This property is exploited by our algorithm which is introduced later.

Theorem 2.5.10 (Poincarè-Bendixson Theorem)
Let $W \subset \mathbb{R}^2$ be an open subset and $v : W \to \mathbb{R}^2$ a two dimensional, continuously differentiable vector field. Let further $L \subset W$ be a nonempty compact limit set of the vector field v that contains no critical point. Then L describes a closed streamline.

Proof:
See [HS74], pages 248 and 249. ❑

Corollary 2.5.11
Let $W \subset \mathbb{R}^2$ be an open subset and $v : W \to \mathbb{R}^2$ a two dimensional, continuously differentiable vector field. Let further $D \subset W$ be a nonempty compact subset which contains no critical point and s a streamline inside D. If the streamline s does not leave D then there exists a closed streamline inside D.

Using this corollary our algorithm to detect closed streamlines can simply integrate a streamline and check during the integration process if it runs into a compact region that is never left. If we find such a region this corollary states that we found a closed streamline.

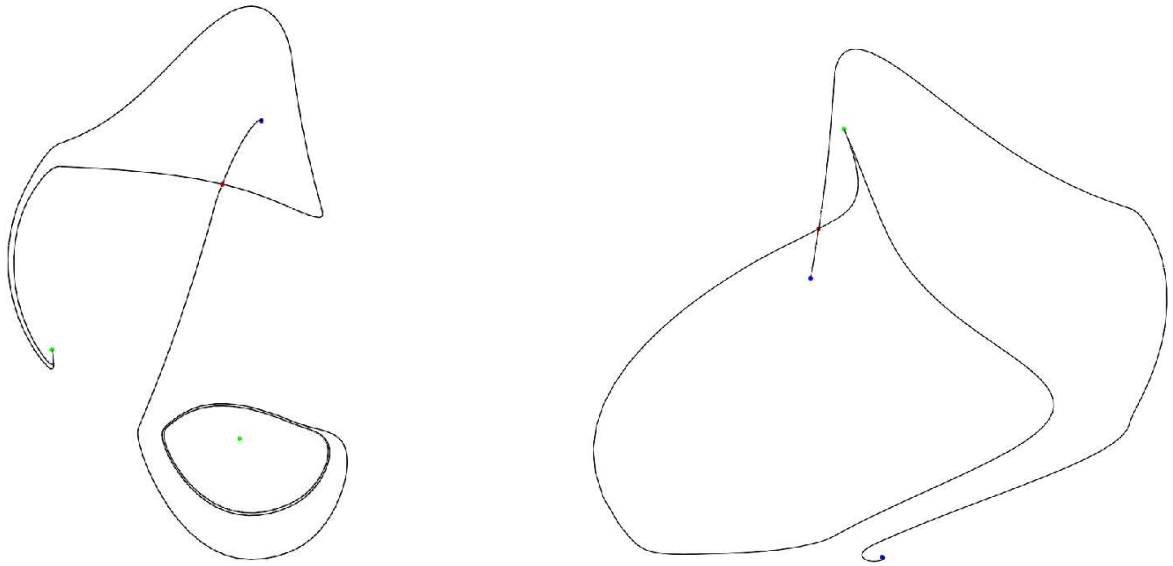
2.6 Vector Field Topology

The topological graph, or simply topology, of a vector field describes the structure of the phase portrait. Considering saddle singularities we can define separatrices.

Definition 2.6.1 (Separatrices)
Let v be a vector field as in definition 2.1.1 and x_0 a saddle singularity. The streamlines emerging in eigendirection are called ***separatrices****.*

Each separatrix connects the saddle point with another critical point or the boundary of the vector field. The separatrices divide the vector field in various topological regions. Each region cannot be left by an individual streamline except for the case where the streamline crosses the boundary. Furthermore, every streamline in that region that does not reach the boundary converges to the same critical point or closed streamline for $t \to \infty$ and to the same critical point or closed streamline for $t \to -\infty$. Now we introduced every concept needed for vector field topology.

Definition 2.6.2 (Topology)
Let v be a vector field as in definition 2.1.1. The topology is built by all critical points, separatrices and closed streamlines of v.



(a) (b)

Figure 2.20: Topological graphs of two vector fields.

Figure 2.20 shows two examples for topological graphs of a simple vector field. The critical points are colored according to its type: saddles are drawn in red, sinks are blue while sources are colored green. The vector field in subfigure (a) contains one closed streamline while the other subfigure does not contain any closed streamlines. In both pictures we can clearly recognize the principle structure of the flow inside the vector fields.

2.7 Structural Stability

If a vector field is slightly perturbed it may happen that the topology stays the same if the change is sufficiently small. This means that there exists a homeomorphism that maps each streamline of the original flow to the perturbed one. This homeomorphism gives a one-to-one correspondence between critical points and closed streamlines of the flow. If such a homeomorphism exists we say that the two flows are *topologically equivalent.*

Definition 2.7.1 (Topologically equivalent)
Let v and w be two vector fields as in definition 2.1.1. Let further ϕ and ψ be the dynamical system according to v respectively w. If there exists a homeomorphism $h : \mathbb{R}^n \to \mathbb{R}^n$ such that for any t_1 there is a t_2 with

$$h(\phi_{t_1}(x)) = \psi_{t_2}(x)$$

then v and w are ***topologically equivalent****.*

To define neighboring vector fields we need a norm on vector fields first. Then we can define neighboring vector fields as vector fields that differ only slightly.

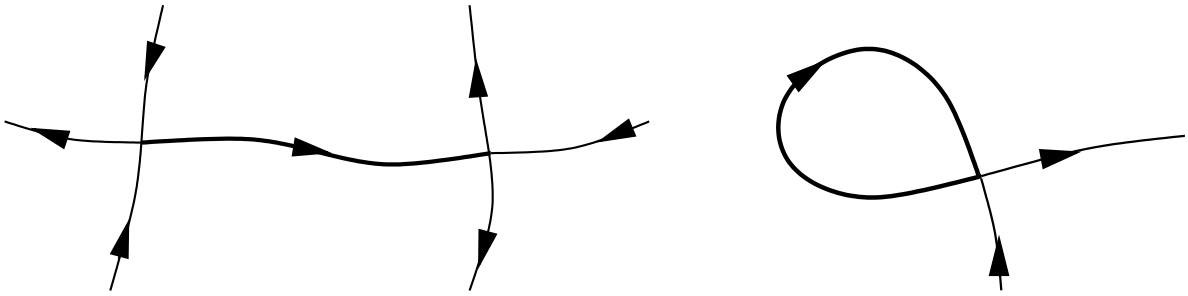
Definition 2.7.2
Let v be vector field as in definition 2.1.1 that is continuous differentiable. Then the ***norm*** *$\|v\|$ of a vector field is defined as $\|v\| = \max(\{\|v(x)\| | x \in W\} \cup \{\|Dv(x)\| | x \in W\})$. We allow $\|v\| = \infty$.*

Definition 2.7.3 (Neighborhood)
Let v be a vector field as in definition 2.1.1 that is continuous differentiable. Let further $N = \{w \in \{v | v : W \to \mathbb{R}^n\} | \|v - w\| < \epsilon\}$. This means that every vector field $w \in N$ is a perturbed version of v. Then N is called a ***neighborhood*** *of v.*

If there exists a neighborhood N of a given vector field v where every vector field is topologically equivalent to the other, the vector field v is called *structural stable.* This means that the topology of the vector field that is slightly perturbed stays the same. The following definition precises that.

Definition 2.7.4 (Structural stable)
Let v be a vector field as in definition 2.1.1. If there is a neighborhood N of v such that every vector field $w \in N$ is topologically equivalent to v then v is called ***structural stable****.*

We now want to give a theorem that explains when a two dimensional vector field is structural stable. But first we need a definition which shows a special configuration concerning saddle singularities. Vector fields containing such a configuration can never be structural stable.



(a) Heteroclinic connection (b) Homoclinic connection

Figure 2.21: Saddle connections.

Definition 2.7.5 (Saddle connections)
Let v be a vector field as in definition 2.1.1 and s_1 and s_2 two saddle singularities of v. If a separatrix connects s_1 and s_2 then this separatrix is called a ***heteroclinic connection****. If a separatrix connects s_1 with itself this separatrix is called* ***homoclinic connection****.*

Figure 2.21 shows the two different configurations. The next theorem shows that for structural stability in a two dimensional vector field it is necessary that the critical points and closed streamlines need to be hyperbolic. Additionally, saddle connections are not allowed.

Theorem 2.7.6
Let $v : W \to \mathbb{R}^2$ a vector field as in definition 2.1.1 with a finite number of critical points and closed streamlines. Then v is structurally stable if and only if

1. *all critical points of v are hyperbolic.*
2. *each closed streamline of v is either repelling or attracting.*
3. *there are no saddle connections.*

Proof:
See [HS74], pages 314 through 317. ❑

2.8 Bifurcations

Closed streamlines are introduced in the field by structural changes. When a vector field changes over time there may be a change in the topology from one state to another.

This, of course, implies that the vector field is not structurally stable in that case. The unstable state in between is called a *bifurcation*. This change may only affect one critical point and its nearer surrounding. Then we call it a *local bifurcation*. The other case is a *global bifurcation* where the global structure of the flow is changed.

Here we consider only bifurcations that result in the creation or vanishing of a closed streamline. The main types are the *Hopf Bifurcation* which is a local bifurcation and the *Periodic Blue Sky in 2D Bifurcation* which is a global one.

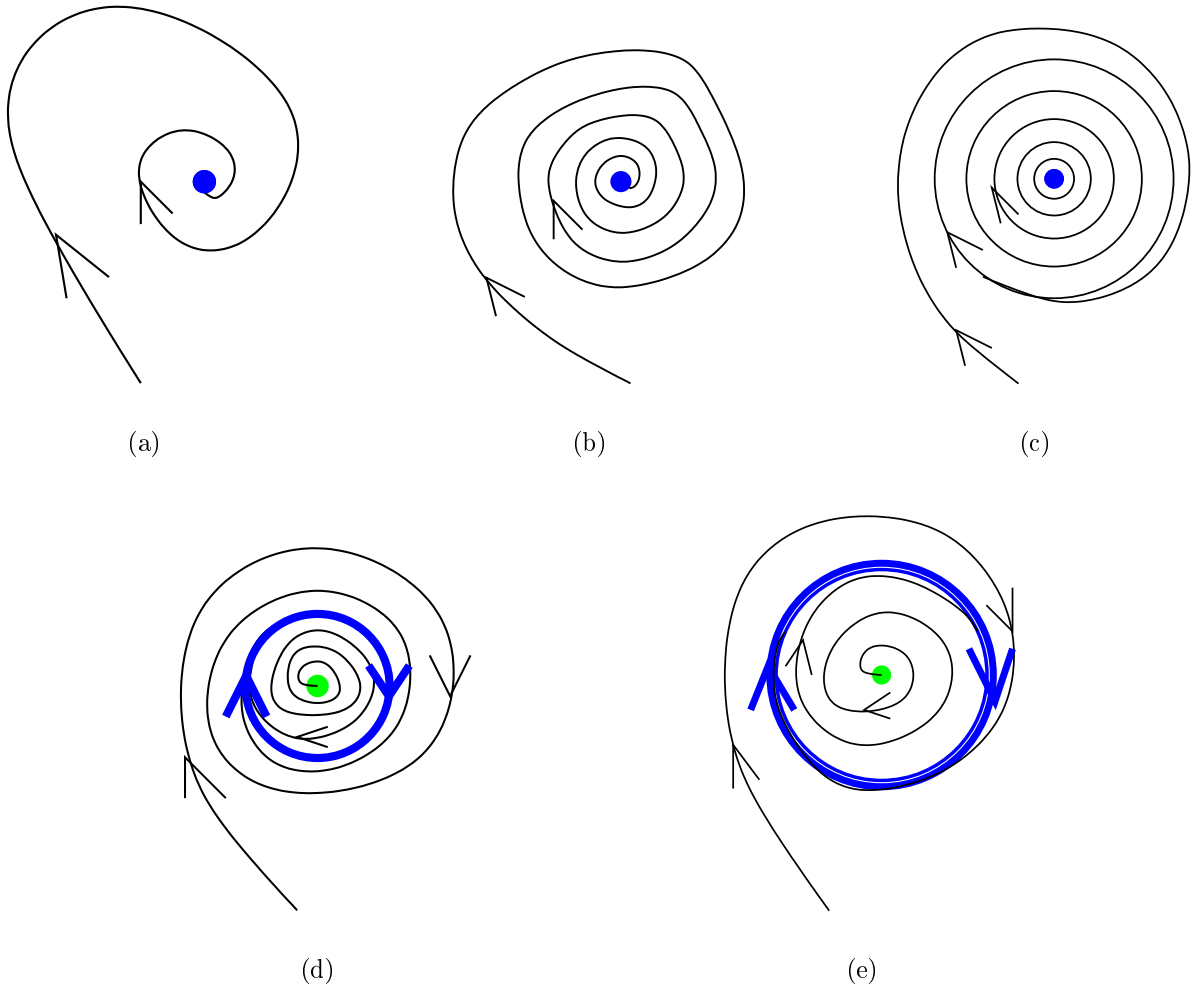
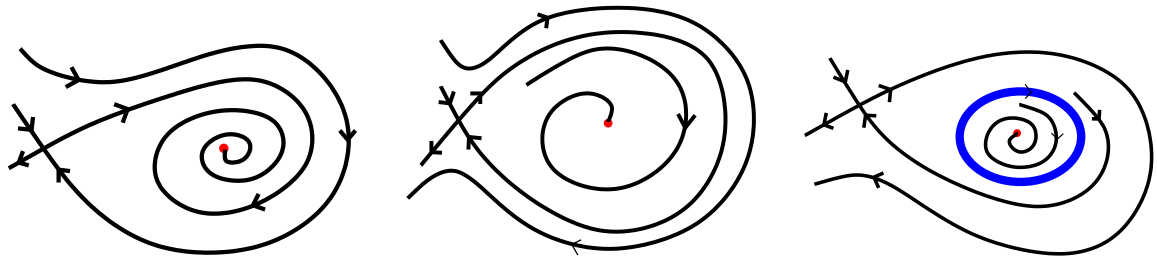


Figure 2.22: Hopf bifurcation.

2.8.1 Hopf Bifurcation

Let us assume that we are given an attracting focus as in figure 2.22a so that a streamline spirals around this critical point and finally converges to it. If the attracting effect weakens the number of rotations of the streamline will increase as in figure 2.22b. Continuing with this process the attracting focus becomes a center point (figure 2.22c) which is an unstable structure: the Hopf bifurcation has occurred. Going further, the structure becomes stable again and we have now a repelling focus. Since the global structure of the flow has not changed, we still have an inflow from the outside and a flow starting at the critical point. Consequently, a closed streamline appears according to the Poincaré-Bendixson-Theorem [GH83] as in figure 2.22d and 2.22e. Inverting the direction of time, we get a transition from a closed streamline with a repelling focus inside into an attracting focus over an instantaneous center where the closed streamline vanishes. Similar transitions are obtained by inverting the direction of the flow, i.e. by replacing sources by sinks. (It may be noted that we can apply the Poincaré-Bendixson-theorem only if the vector field is continuous. Further we have a region without critical points.)

2.8.2 Periodic Blue Sky in 2D Bifurcation



(a) (b) (c)

Figure 2.23: Periodic Blue Sky in 2D.

In this type of bifurcation there are two different types of critical points involved: a saddle and an attracting focus. Figure 2.23a shows the situation. As the attracting effect of the focus gets weaker and weaker we see a homoclinic connection after some time where the saddle is connected to itself as shown in figure 2.23b. This results in a bifurcation: when this configuration breaks up again we find a limit cycle which simply appears out of the blue. The reason for the occurrence of the closed streamline is that the attracting focus is totally unaffected by the whole event. Since there is an outflow to the critical point inside and to the saddle there must be a critical point or a closed streamline in this region according to the Poincaré-Bendixson theorem. Because of the fact that

there are only the two critical points a closed streamline emerged. This configuration is shown in figure 2.23c. Other bifurcations of the same type can be constructed by inverting time or replacing the attracting focus with a repelling one.

Chapter 3

State of the Art

Flows occur in various different forms in science and engineering. For instance, a wind tunnel experiment results in such a flow. The path of the air describes how the flow behaves around a special object as, for example, a car. Inflows into special parts, like a thrust chamber, are of interest also. There, the flow describes the injection of the gas. The composition of gas and oxygen is very important in combustion processes. A better insight into the flow can help optimizing this process.

Several visualization methods are available at present. Here, we concentrate on describing these methods that are useful in our application area. An overview over the various visualization methods can also be found in other publications [GLW97] and PhD theses [Löf98][Tri02].

3.1 Vector Field Visualization

Various methods exist that show different aspects of vector fields. Hedgehog methods [PvW93] draw arrows tangential to the flow. Each arrow represents a vector at that position. The length shows the velocity. The principle structure of the flow can be recognized using this method. But special features like closed streamlines can easily be overseen. In the three dimensional case, occlusion problems occur so that an analysis of the vector field is difficult with this method.

Texture based methods visualize the whole phase portrait. There are mainly two different methods for creating the texture: spot noise [vW91][dLvW95][dLPV96] and line integral convolution (LIC) [CL93]. To create a spot noise texture, randomly weighted and positioned spots are accumulated. The shape of the spots controls the texture locally. If we align, for instance, the larger axis of the spots parallel to the flow direction the resulting texture visualizes the vector field. The LIC method uses a white noise texture as a basis. This texture gets smeared in the flow direction: another texture is created where for every pixel a short streamline is computed and the color values of

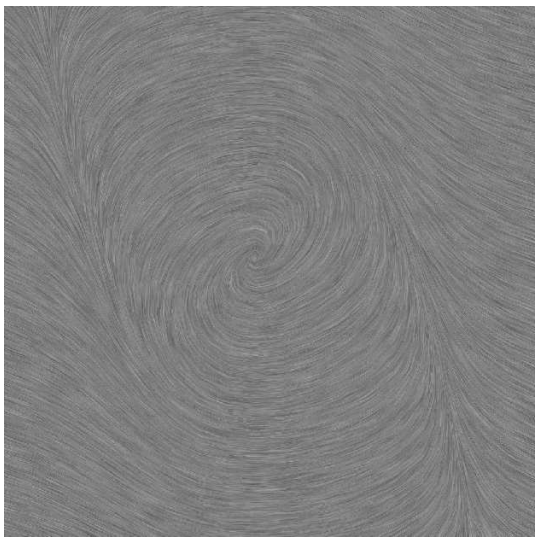


Figure 3.1: Vector field containing a closed streamline visualized using the LIC method.

each pixel of the white noise texture that is crossed by this streamline is summed up. Figure 3.1 shows an example of a LIC image.

Many extensions [KB96][SJM96] and performance optimizations [SH95][SZH96] exist for this method. To introduce orientation information oriented line integral convolution (OLIC) was proposed by Wegenkittl et al. [WG97][WGP97]. For time-dependent flows the standard method is not suitable because it results in a flickering animation. Therefore some extensions exist [Lan93][FC95] like for instance unsteady flow line integral convolution (UFLIC) [SK97][SK98]. This method is based on the fast LIC algorithm [SH95]. The difference is in the convolution kernel: to achieve temporal coherence only the pixel calculations with a smaller time-stamp than the actual one are considered.

To track a particle in the flow over time streamlines, streaklines, and pathlines[Han93] [Lan94] are used. A streamline shows the path of a massless particle in the flow. Such a particle follows the trajectory of the dynamical system. A streakline visualizes the path of dye injected for a period of time at a fixed position into a time dependent flow while a pathline only follows a single particle. A particle corresponds to a point moving through the flow. If we use more general objects like lines, circles, or implicit surfaces streamsurfaces, streamribbons, streamtubes, or streamballs are created [BDH+94]. Also, an n-sided polygon can be placed perpendicular to the flow and moved along the trajectory [SVL91]. This method additionally depicts local flow attributes, like rotation and shear.

3.2 Topological Methods

Topological methods depict the structure of the flow by connecting sources, sinks, and saddle singularities with separatrices. Critical points were first investigated by Perry [PF74][Per84][PC87], Dallmann [Dal83], Chong [CPC90] and others. The method itself was first introduced in visualization for two dimensional flows by Helman and Hesselink [HH89b][HH89a][HH90][HH91][Hel97]. Several extensions to this method exist. Scheuermann et al. [SHJK00] extended the method to work on a bounded region. To get the whole topological skeleton of the vector field, points on the boundary have to be taken into account, also. These points are called boundary saddles. To create a time dependent topology for two dimensional vector fields, Helman and Hesselink [HH91] use the third coordinate to represent time. This results in surfaces representing the evolution of the separatrices. A similar method is proposed by Tricoche et al. [TSH01][TWSH02] but this work focuses on tracking singularities through time. Although closed streamlines can act in the same way as sources or sinks, they are ignored in the considerations of Helman and Hesselink and others.

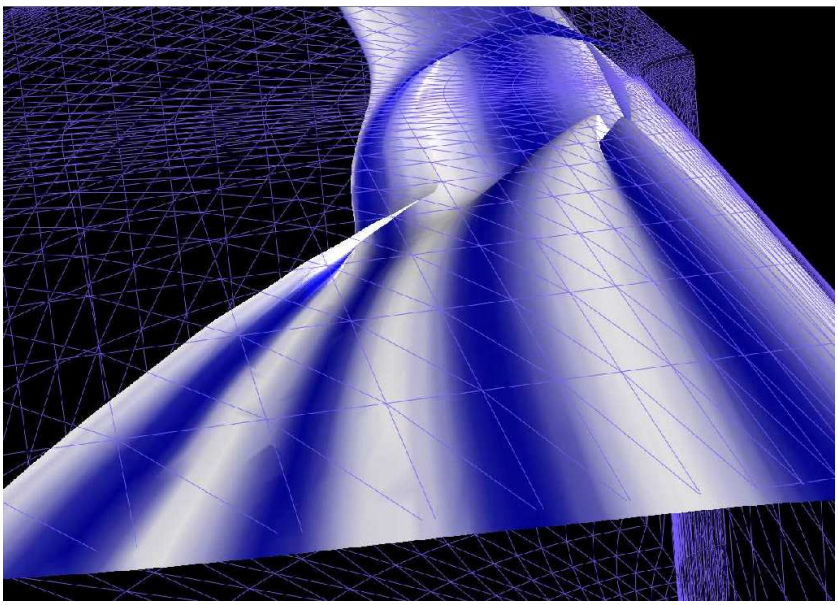


Figure 3.2: Streamsurface inside the blunt fin dataset from NASA [HB90].

To extend this method to three dimensional vector fields, Globus et al. [GLL91] presented a software system that is able to extract and visualize some topological aspects of three dimensional vector fields. The various critical points are characterized using the eigenvalues of the Jacobian. This technique was also suggested by Helman and Hesselink [HH91]. But the whole topology of a three dimensional flow is not yet available. There, streamsurfaces are required to represent separatricies. A few algorithms for computing

streamsurfaces exist [Hul92][SBH+01] but are not yet integrated in a topological algorithm. Figure 3.2 shows a streamsurface inside the famous blunt fin dataset provided by NASA [HB90] constructed with the algorithm by Scheuermann et al. [SBH+01].

3.3 Closed Streamlines in Visualization

There are some algorithms to find closed streamlines in dynamical systems that can be found in the numerical literature. Aprille and Trick [AT72] proposed a so called shooting method. There, the fixed point of the Poincaré map is found using a numerical algorithm like Newton-Raphson. Dellnitz et al. [DJ99] detect almost cyclic behavior. It is a stochastical approach where the Frobenius-Perron operator is discretized. This stochastical measure identifies regions where trajectories stay very long. But these mathematical methods typically depend on continuous dynamical systems where a closed form description of the vector field is available. This is usually not the case in visualization and simulation where the data is given on a grid and interpolated inside the cells. Van Veldhuizen [vV87] uses the Poincaré map to create a series of polygons approximating an attracting closed streamline. The algorithm starts with a rough approximation of the closed streamline. Every vertex is mapped by the Poincaré map iteratively to get a finer approximation. Then, this series converges to the closed streamline.

To get a hierarchical approach for the visualization of invariant sets, and therefore closed streamlines also, Bürkle et al. [BDJ+99] enclose the invariant set by a set of boxes. They start with a box that surrounds the invariant set completely. This box is successively bisected in cycling directions. It is always ensured that the result still includes the complete invariant set. Using this bisection, an approximation of the invariant set is finally found which can be rendered using a volume renderer. The publication of Guckenheimer [Guc00] gives a detailed overview concerning invariant sets in dynamical systems.

Some publications deal with the analysis of the behavior of dynamical systems. Schematic drawings showing the various kinds of closed streamlines can be found in the books of Abraham and Shaw [AS84][AS88]. Fischel et al. [FDM+97] presented a case study where they applied different visualization methods to dynamical systems. In their applications also strange attractors, like the Lorentz attractor, and closed streamlines occur. So called sweeps which are trajectories represented as tubes are used. These sweeps allow to introduce a color coding scheme. For instance, the color can help to recognize that a trajectory still slowly moves towards a closed streamline that weakly attracts.

Wegenkittl et al. [WLG97] visualize higher dimensional dynamical systems. To display trajectories parallel coordinates [ID90] are used. A trajectory is sampled at various points in time. Then these points are displayed in the parallel coordinate system and a surface is extruded to connect these points. As an example, also a chaotic attractor

derived from the Lorentz system is visualized. Hepting et al. [HDER95] study invariant tori in four dimensional dynamical systems by using suitable projections into three dimensions to enable detailed visual analysis of the tori. This visualization can help when limits of mathematical analysis are reached to get more insight into the dynamical system.

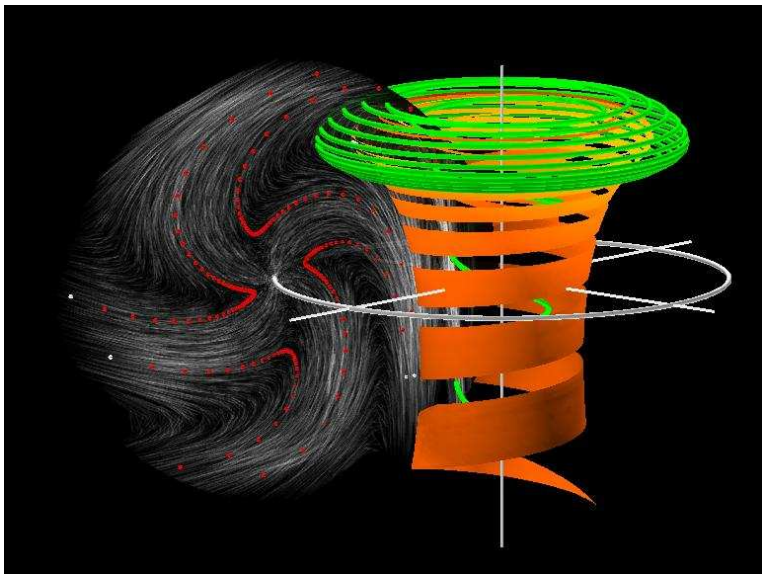


Figure 3.3: Poincaré section with closed streamline (image courtesy of Helwig Hauser, VRVis[LKG97]).

Löffelmann [Löf98][LKG97] uses Poincaré sections to visualize closed streamlines and strange attractors. Poincaré sections define a discrete dynamical system of lower dimension which is easier to understand. The Poincaré section which is transverse to the closed streamline is visualized as a disk. On the disk, spot noise is used to depict the vector field projected onto that disk. By this method, it can be clearly recognized whether the flow, for instance, spirals around the closed streamline and is attracted or repelled or if it is a rotating saddle. Additionally, streamlines and streamsurfaces show the vector field in the vicinity of the closed streamline that is not located on the disk visualizing the Poincaré section. Figure 3.3 shows an example of that visualization method.

3.4 Distributed Computing

Due to increasing computing power during the last years flow simulations became larger and larger at finer resolutions. Often, these simulations are computed on a parallel machine. Consequently, it takes a long time to compute an appropriate visualization for

such big datasets. Especially, when dealing with an algorithm that needs to compute many streamlines it helps a lot to compute this in parallel also. Several parallel algorithms exist in visualization. In the following, we want to list a few of them that deal with problems that are related to this work.

Sujudi et al. [SH96] presented a method for computing streamlines in a parallel environment by splitting the dataset into several sub-domains. If the streamline leaves a sub-domain another process responsible for the actual domain has to continue the computation. Reinhard et al. [RCJ99] proposed a parallel rendering method that distributes tasks for each ray which has to be computed to the different processors of the parallel machine. A parallelization of line integral convolution was presented by Zöckler et al. [ZSH96] where the vector field is divided into several subdomains depending on the number of processors used.

Chapter 4

Detection and Visualization in Planar Flows

This chapter describes an algorithm that detects if an arbitrary streamline c converges to a closed curve, also called a limit cycle. This means that c has γ as α- or ω-limit set depending on the orientation of integration. We do not assume any knowledge on the existence or location of the closed curve, so that the algorithm can detect stable closed streamlines. We exploit the fact that we use linear interpolation inside the cells for the proof of our algorithm. But the principle of the algorithm works on any piecewise defined planar vector field where one can determine the topology inside the pieces. First, we describe how to explain and prove the presence of a closed streamline and finally we give a procedure how to find the exact position of the closed streamline.

4.1 Detection of Closed Streamlines

In a precomputational step every singularity of the vector field is determined. To find all stable closed streamlines we mainly compute the topological skeleton of the vector field. We use an ordinary streamline integrator, like for instance an ODE solver using Runge-Kutta. But we extended this streamline integrator so that it is able to detect closed streamlines. In order to find all closed streamlines that reside inside another closed streamline we have to continue integration after we found a closed streamline inside that region.

4.1.1 Theory

The basic idea of our streamline integrator is to determine a region of the vector field that is never left by the streamline. According to the Poincaré-Bendixson-Theorem, a streamline approaches a closed streamline if no singularity exists in that region.

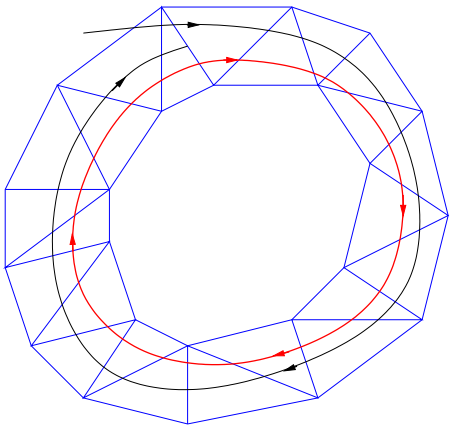


Figure 4.1: A streamline approaching a limit cycle has to reenter cells.

Notation 4.1.1 (Actually investigated streamline)
We use the term ***actually investigated streamline*** *to describe the streamline that we check if it runs into a limit cycle.*

To reduce computational cost we first integrate the streamline using a Runge-Kutta-method of fifth order with an adaptive stepsize control. Every cell that is crossed by the streamline is stored during the computation. If a streamline approaches a limit cycle it has to reenter the same cell again as shown in figure 4.1. This results in a *cell cycle*:

Definition 4.1.2 (Cell cycle)
Let s be a streamline in a given vector field v. Further, let G be a set of cells representing an arbitrary rectangular or triangular grid without any holes. Let $C \subset G$ be a finite sequence $c_0, \ldots, c_n$ of neighboring cells where each cell is crossed by the streamline s in exactly that order and $c_0 = c_n$. If s crosses every cell in C in this order again while continuing, C is called a ***cell cycle****.*

This cell cycle identifies the region mentioned earlier. To check if this region can be left we could integrate backwards starting at every point on the boundary of the cell cycle. If there is one point converging to the actually investigated streamline we know for sure that the streamline will leave the cell cycle. If not, the actually investigated streamline will never leave the cell cycle. Since there are infinitely many points on the boundary this, of course, results in a non-terminating algorithm. To crack this problem we have to reduce the number of points we have to check. Therefore we define *potential exit points*:

Definition 4.1.3 (Potential exit points)
Let C be a cell cycle in a given grid G as in Definition 4.1.2. Then there are two kinds of ***potential exit points****. First, every vertex of the cell cycle C is a* ***potential exit***

***point**. Second, every point on an edge at the boundary of C where the vector field is tangential to the edge is also a **potential exit point**. Here, only edges that are part of the boundary of the cell cycle are considered. Additionally, only the **potential exit points** in the spiraling direction of the streamline need to be taken into account.*

To determine if the streamline leaves the cell cycle we start a backward integrated streamline to see where we have to enter the cell cycle in order to leave it at that exit. We will show later that it is sufficient to only check these potential exit points if we want to figure out if the streamline can leave the cell cycle.

Notation 4.1.4 (Backward integrated streamline)
*We use the term **backward integrated streamline** for the streamline we integrate by inverting the vectors of the vector field starting at a potential exit point in order to validate this exit point.*

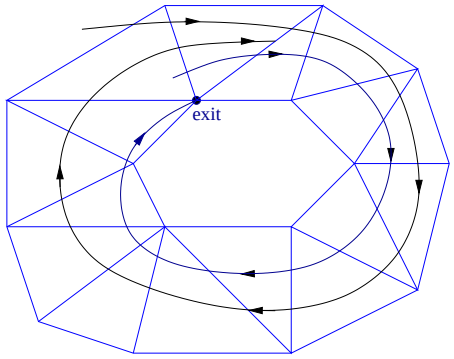


Figure 4.2: If a real exit point can be reached, the streamline will leave the cell cycle.

Definition 4.1.5 (Real exit points)
*Let P be a potential exit point of a given cell cycle C as in definition 4.1.3. If the backward integrated streamline starting at P does not leave the cell cycle after one full turn through the cell cycle, the potential exit point is called a **real exit point**.*

Since a streamline cannot cross itself the backward integration starting at a real exit point converges to the actually investigated streamline. Consequently, the actually investigated streamline leaves the cell cycle near that real exit point. Figure 4.2 shows such a real exit point.

If on the other hand no real exit point exists we can determine for every potential exit point where we have a region with an inflow that leaves at that potential exit. Consequently, the actually investigated streamline cannot leave near that potential exit point.

With these definitions we can formulate the main theorem for our algorithm:

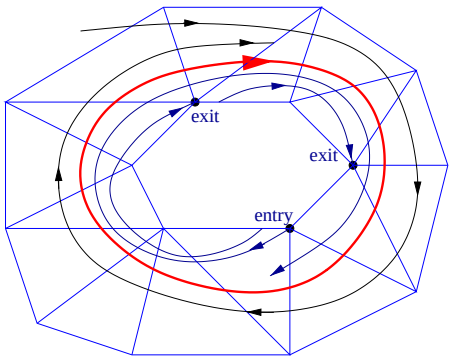


Figure 4.3: If no real exit point can be reached, the streamline will approach a limit cycle.

Theorem 4.1.6
Let C be a cell cycle with no singularity inside and E the set of potential exit points. If there is no real exit point among the potential exit points E or there are no potential exit points at all then there exists a closed streamline inside the cell cycle.

Proof: *(Sketch)*
Let C be the cell cycle. It is obvious that we cannot leave the cell cycle C if all backward integrated streamlines started at every point on the boundary of C leave the cell cycle C. According to the Poincaré-Bendixson-theorem, there exists a closed streamline inside the cell cycle in that case.

We will show now that it is sufficient to treat only the potential exit points. If the backward integrated streamlines starting at all these potential exit points leave the cell cycle the backward integration of any point on an edge will also do.

Figure 4.4 shows the different configurations of potential exits. Let E be an arbitrary point on an edge between two potential exit points. In part *(a)* both backward integrated streamlines starting at the vertices V_1 and V_2 leave the cell cycle. Consequently, E cannot be an exit. It would need to cross one of the other backward integrated streamlines which contradicts with theorem 2.1.8.

Part *(b)* of figure 4.4 shows the case where the vector at a point on the edge is tangential to the edge. Obviously, if E lies between V_1 and T the backward integrated streamline will leave the cell cycle immediately. If it lies between T and V_2 and converges to the actually investigated streamline it has to cross the backward integrated streamline started at T. This contradicts with theorem 2.1.8. Because of the linear interpolation at the edge, part *(c)* is also impossible.

We have shown that the actually investigated streamline cannot leave the cell cycle. Consequently, there exists a closed streamline inside the cell cycle C since there is no singularity inside C. ❑

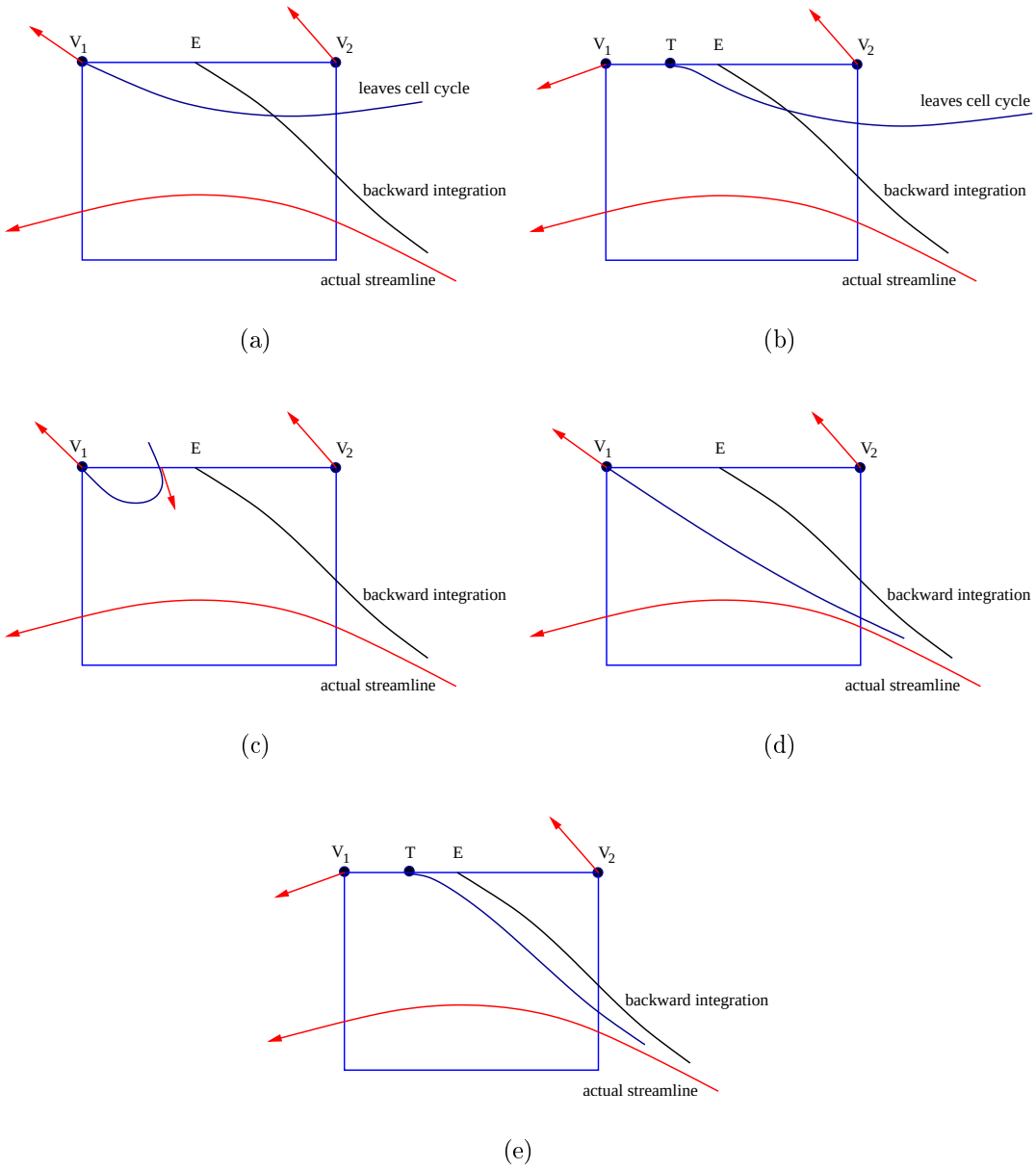


Figure 4.4: Different cases of potential exits. (a) and (b) is impossible because streamlines cannot cross each other, (c) contradicts with the linear interpolation on an edge, in (d) and (e) both backward integrations converge to the actual streamline so that the point E is a real exit.

Remark 4.1.7
To get a possible configuration the backward integration starting at the vertex V_1 must also converge to the streamline because it cannot cross the backward integration starting at point E as in part (d) *of figure 4.4. Part* (e) *explains why we also need to investigate the tangential case. If we start a backward integrated streamline at point E it converges towards the actually investigated streamline. But if we only consider the vertices of the edge, both exit points may be no real exit points. Therefore we also have to start a backward integrated streamline at the point T, where the vector field is tangential to the edge, to figure out that we leave the cell cycle at this edge. On the other hand, a backward integrated streamline starting at any point between V_1 and T immediately leaves the cell cycle due to the linear interpolation.*

4.1.2 Algorithm

With theorem 4.1.6 we are able to describe our algorithm in detail. It mainly consists of the same three different states:

❶ streamline integration: identifying one cell change after the other, check at each cell if we complete a cell cycle.

❷ checking for exits: going backwards through the crossed cells and looking for potential exit points.

❸ validating exit: integrating backwards a curve from potential exit through the whole cell cycle.

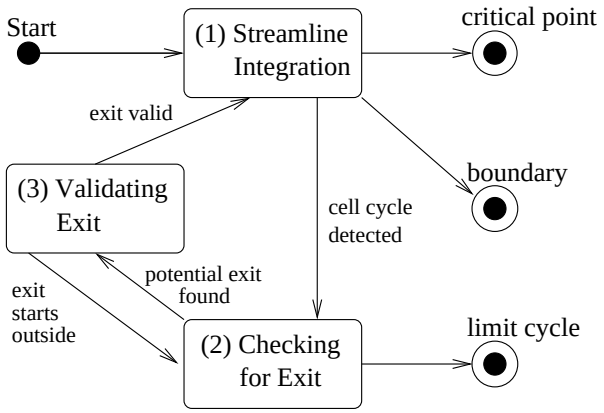


Figure 4.5: The UML state diagram of our algorithm.

The algorithm switches its states after the events shown in the state diagram in figure 4.5. We use a standard integration method to compute the streamline, first. In

this step we only check for cell cycles. This saves computational time since the checking of all the exits is rather expensive. If we detect a cell cycle we have to find all potential exit points. After that we need to validate each of the potential exit points to figure out if there is a real exit point among them. If this is the case we did not run into a closed streamline yet. Therefore we continue with the standard integration. The algorithm exits if we could not find a real exit point among all the potential exit points or if we reached a critical point or the boundary of the vector field.

Remark 4.1.8
Theorem 4.1.6 guarantees that our algorithm detects closed streamlines if we check every potential exit point.

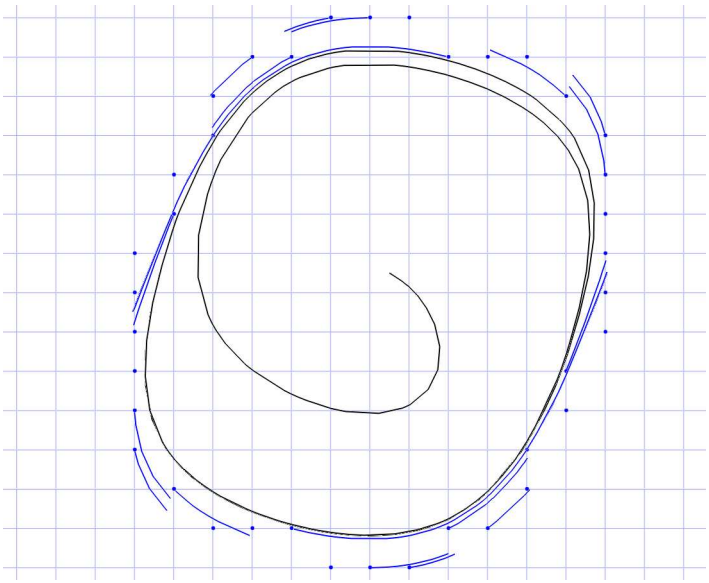


Figure 4.6: Exits of a cell cycle.

Figure 4.6 shows a real example of our algorithm. There we start a streamline near the source in the center of the figure. This streamline spirals until we find the first cell cycle. We stopped the integration there for this example. The figure also shows all exits and its backward integrated streamlines. The streamline itself is colored black. The grid is displayed in a lighter color. In this example, every potential exit point is shown. We can see that potential exit points which are passed by a backward integrated streamline do not necessarily need to be investigated because if the backward integrated streamline leaves the cell cycle the other one will also do. Figure 4.7 shows this in detail. There the backward integrated streamline starting at *Exit 2* also has to leave the cell cycle because it cannot cross the backward integrated streamline starting at *Exit 1*. In the

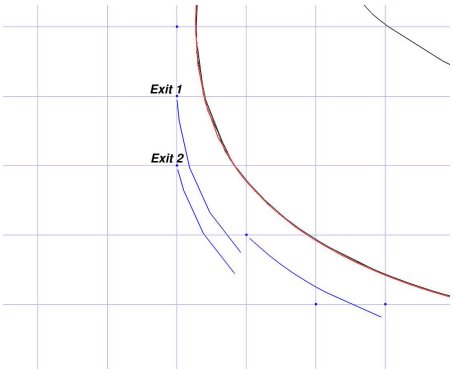


Figure 4.7: Exit of the cell cycle which does not need to be investigated.

other case, where the backward integrated streamline started at *Exit 1* stays inside the cell cycle, we have to continue the actually investigated streamline, anyway.

Since the streamline spirals from the inner region to the outside, we only have to consider the potential exits in that direction. In the example, every backward integrated streamline leaves the cell cycle. Consequently, there is a limit cycle in this cell cycle which can be localized as described in the next section.

4.2 Exact Location of Closed Streamlines

Since we know a region that is never left by the streamline we can find the exact position of the closed streamline using the Poincaré map. This map is described in detail in the subsection 2.5.2.

To find the exact position of the closed streamline we can use the edge where we detected the cell cycle as a Poincaré section. Then we only have to find the fixed point of the Poincaré map. We use a binary search to find this fixed point: we divide the edge where we detected the cell cycle into two parts. At the mid point we start a streamline to see which part of the edge is intersected by the streamline after one full turn. Since the streamline cannot leave the cell cycle, it is guaranteed that the streamline intersects one part of the edge. Then, this part is subdivided again and we start another streamline at the mid point. This process continues until we are close enough to the fixed point of the Poincaré map. We use the length of the part of the edge as a stopping criterion.

This fixed point gives us a point lying on the closed streamline. If we start another streamline at that point this streamline will follow the closed streamline we are looking for. After one turn, i.e. after reaching the start point again, we know the exact location of the closed streamline.

4.3 Results

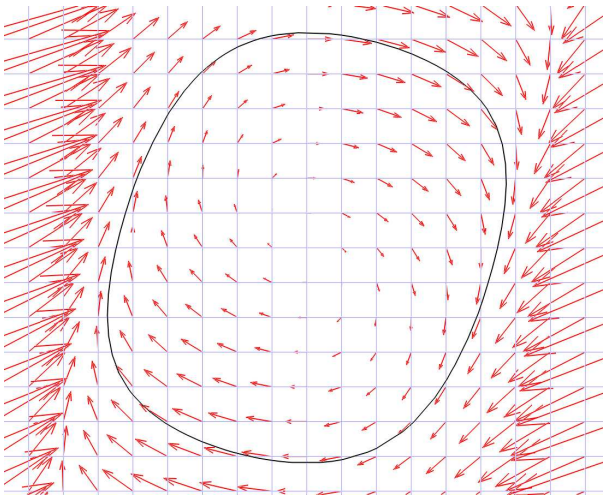


Figure 4.8: Simple vector field with limit cycle.

The first example is a vector field that contains only one closed streamline. It is sampled on a regular grid using a slightly changed *Van der Pol's equation*. The defining equation for the vector field V is

$$V\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} y - x^3 + \mu x \\ -x \end{pmatrix} \quad . \tag{4.1}$$

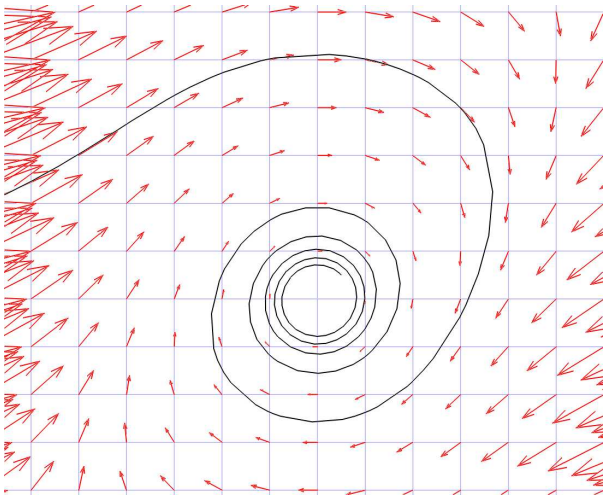


Figure 4.9: Simple vector field with no limit cycle.

According to Hirsch and Smale[HS74] a limit cycle occurs if we set $0 < \mu \leq 1$. An analysis of the field shows that we have a source at $(0, 0)$. When starting our algorithm near that singularity it integrates the streamline until it detects the limit cycle as shown in figure 4.8. Figure 4.8 also includes the hedgehog of the vector field, a glyph visualization method where we use arrows representing the vectors at the corresponding position. The arrows are twice as long as the vectors of the field.

In figure 4.9, we investigate a vector field which spirals from the singularity to the outer regions. Again, we used equation 4.1 but we set $\mu = -0.02$ to compute the vector field. Consequently, there is no limit cycle in the vector field. Our algorithm correctly fails to detect one, when started near the singularity at $(0, 0)$ and continues the streamline computation until it reaches the boundary of the vector field. Here, also the hedgehog of the vector field is displayed scaled by a factor of two.

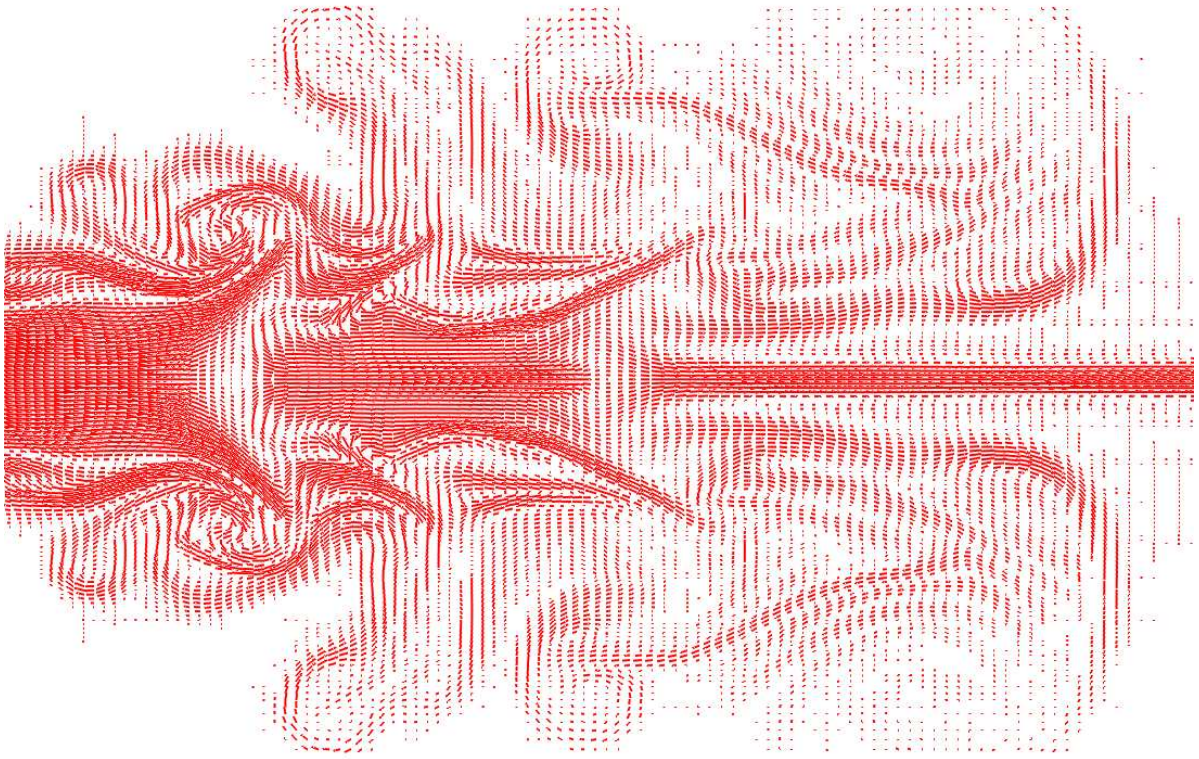


Figure 4.10: Vorticity vector field of a turbulent flow – hedgehog.

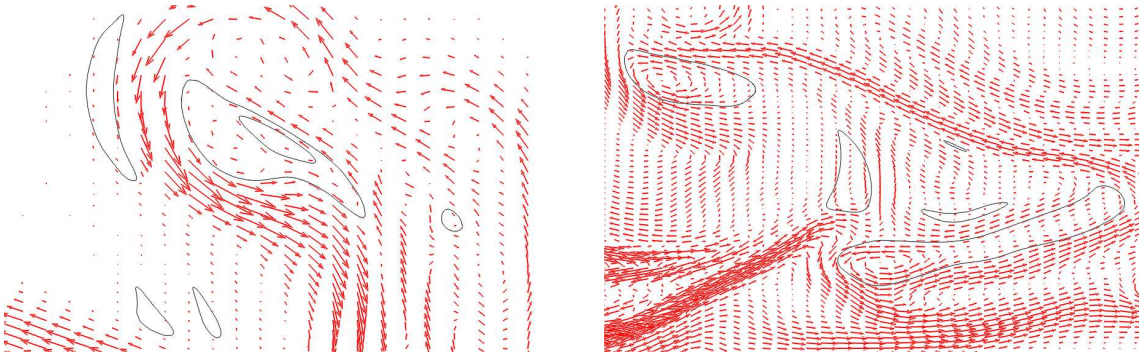


Figure 4.11: Vorticity vector field of a turbulent flow – limit cycles.

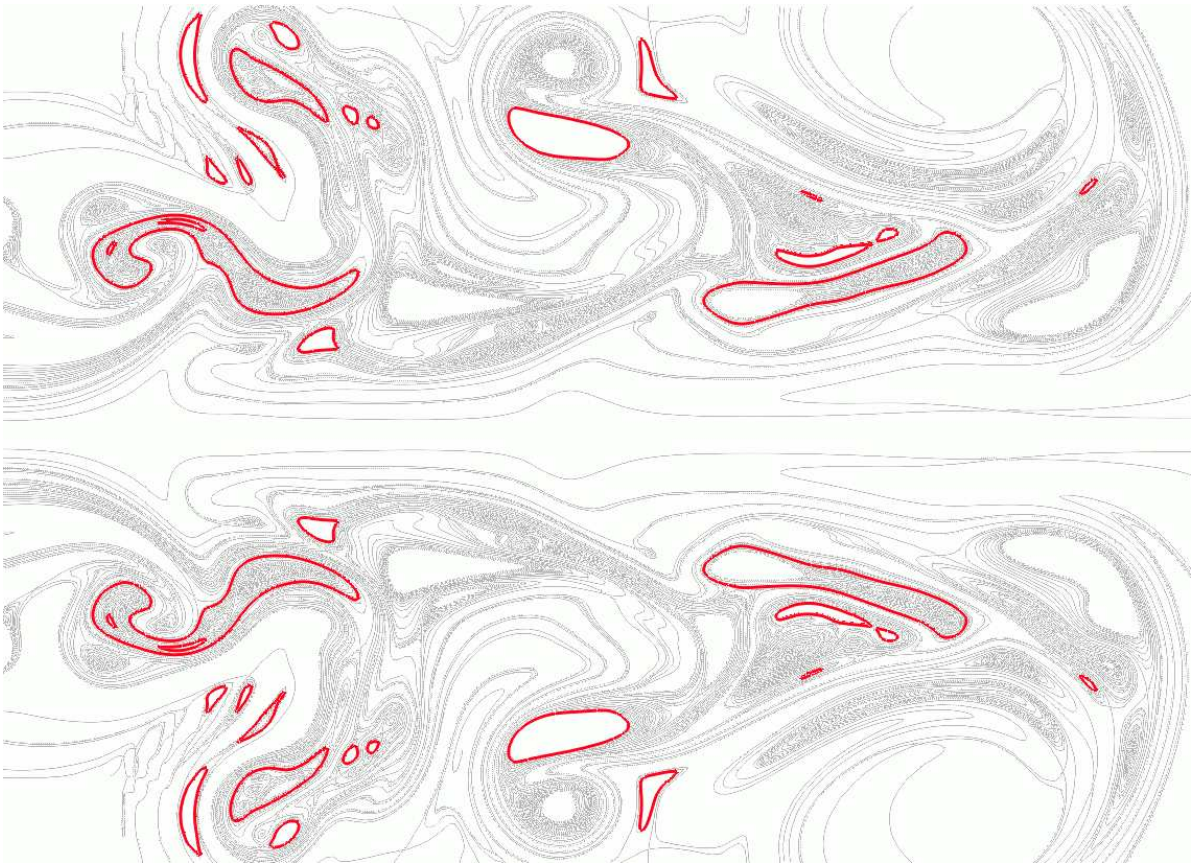


Figure 4.12: Vorticity vector field visualized by the topological skeleton including closed streamlines.

The third example is a simulation of a swirling jet with an inflow into a steady medium. The simulation originally resulted in a three dimensional vector field but we used a cutting plane and projected the vectors onto this plane to get a two dimensional field. This dataset was provided by Prof. Kollmann from the mechanical engineering department at the University of California at Davis. In this application one is interested in investigating the turbulence of the vector field and in regions where the fluid stays very long. This is necessary because some chemical reactions need a special amount of time. These regions can be located by finding closed streamlines. Figure 4.10 shows the hedgehog of that vector field scaled by a factor of two. In figure 4.11 one can see some of the closed streamlines detected by our algorithm. All these limit cycles are located in the upper region of the vector field. Additionally figure 4.11 includes the hedgehog where the arrows representing the vectors are four times longer than the corresponding vector. Figure 4.12 shows all closed streamlines of this vector field including the topological skeleton.

To compare our enhancements with usual streamline computation methods, we implemented an algorithm which computes the topological skeleton as described in [HH91]. Therefore we have to determine the singularities. Then we start a streamline at each saddle point displaced a little bit in positive and negative eigendirection of both eigenvectors. Remind that our algorithm does not need any exit conditions other than the detection of closed streamlines or reaching a singularity or the border of the data!

To get an idea of the computational cost of our method we also implemented a simple ODE solver to compute the streamlines. The vector field shown in figure 4.10 contains 337 singularities. The algorithm using a simple ODE solver needed 738 seconds to compute the topological skeleton on a Pentium II 350 MHz. Using our streamline integration method, which uses the same ODE solver but checks for limit cycles, we only needed 604 seconds on the same system which is 18 percent faster! The reason for that is that we do not need to wait until the ODE solver reaches a certain number of steps if we run into a limit cycle. This saves some time which we can use to check for limit cycles.

4.4 Limitations

If more than one closed streamline crosses the same cell, the algorithm may fail to detect these closed streamlines. For instance, there is a structural unstable configuration with one closed streamline inside the other. One closed streamline acts like a source, lets say the inner one, while the other one behaves like a sink so that the flow starts at the first and is attracted by the second one. Since there is an outflow out of the cell the algorithm cannot distinguish between a regular outflow and this configuration.

Figure 4.13 shows an example for such a configuration. The flow direction inside the first closed streamline is the same than behind the second one. It looks the same as if

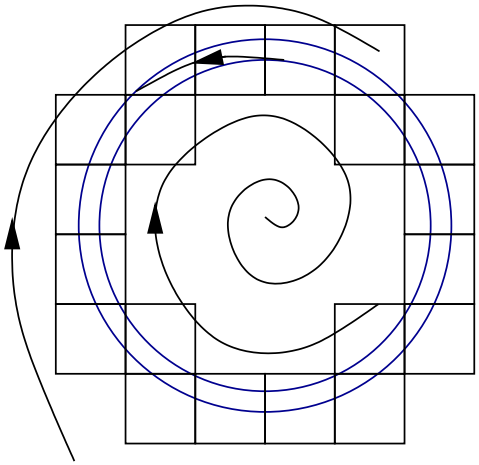


Figure 4.13: Unstable closed streamlines.

there are no closed streamlines at all. Consequently, the algorithm fails to detect both closed streamlines.

Department of Computer Science, University of Kaiserslautern, Germany

Chapter 5

Parallel Detection of Closed Streamlines

To determine the closed streamlines of a vector field many streamlines have to be computed. In fact, we compute the topological skeleton. This graph leads us to the closed streamlines. Since the number of streamlines may be large depending on the given vector field, this may take several minutes or even hours, especially since we also have to compute even more backward integrated streamlines. Therefore we created a parallel version of this algorithm to decrease computational time by distributing the streamline computation to several clients. Some more information on concurrent programming can be found in the literature [Sch97][Mul93] [Ung97][Aga89].

First, we describe some parallel machines that can be used for our algorithm. Then two different parallelization methodologies are discussed in the next section. In the end of this chapter we show the results including different timings on several test systems.

5.1 Parallel Machines

In this section we describe briefly some parallel machines, the Cray/SGI T3E, the IBM RS/6000 SP, and Linux clusters. The first one uses a *distributed shared memory* concept while the other two ones do not share their memory at all.

5.1.1 Cray/SGI T3E

The Cray/SGI T3E is available since 1996. It is a distributed shared memory system where every node shares its memory with all the others. It uses a virtual address space to access the memory that is spread among all nodes. The processor used for the nodes is the DEC Alpha processor 21164. This processor consists of two integer and two floating point units with IEEE 64 bit arithmetic. It has an eight KB first level and 96 KB second

level cache directly on the chip.

The *GigaRing* technology based on the IEEE SCI standard is used to connect the nodes. Every node is bi-directional connected to its neighbor in a three dimensional network topology.

5.1.2 IBM RS/6000 SP

The IBM RS/6000 SP uses POWER4 microprocessors. This type of processor has an *SMP-on-a-chip* design. It consists of two 1.3 GHz processors including second level cache directly on one chip. Every node has its own memory. So the parallel program has to use a message passing system as for instance PVM or MPI.

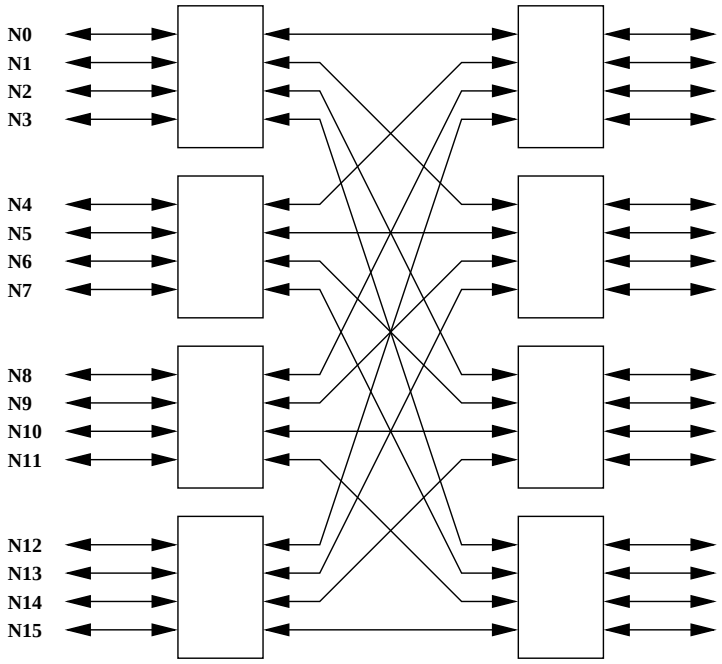


Figure 5.1: HPS basic element of an IBM RS/6000 SP for 16 nodes.

Up to sixteen nodes are grouped together in a network configuration as shown in figure 5.1 using *high performance switches* (HPS). If more than sixteen nodes are used several of these groups have to be interconnected.

5.1.3 Linux Clusters

With Linux clusters there are no restrictions concerning network topology, memory, or CPU speed. Almost every standard PC component can be used in a Linux cluster. Even several desktop Linux computers that are connected through an Ethernet can be

called a Linux cluster. But usually the network is the bottleneck in such a system. Therefore faster network devices like for instance a GigaBit network device or Myrinet host interface is used. A flat, tree shaped network topology is possible for a Linux cluster. But especially with a greater number of nodes a network with routes from any to any other node is desirable to avoid collisions and facilitate faster transfers.

Because of their low prices and the great scalability Linux clusters become more and more popular. They also appear nowadays in the list of the top 500 Supercomputer Sites.

5.1.4 Comparison

	Cray/SGI T3E	IBM RS/6000 SP	Linux Cluster
Processor type	DEC Alpha 21164	p690 server (dual)	e.g. Athlon
Number of processors	up to 2176	up to 16	unlimited
Memory size (per node)	512 MB	–	up to 3 GB
Clock speed (per node)	up to 675 MHz	up to 1.3 GHz	up to 2GHz
Network bandwidth	500 MB/s	500 MB/s	up to 250 MB/s
Peak Performance	3 TFLOPS	2.6 TFLOPS	unlimited

Figure 5.2: Technical specifications of different parallel machines.

The main advantage of Linux clusters is the low price of standard PC components. It is very extendable because there is no limit in the number of nodes used in the cluster. In principle, you only have to add a new computer to the network to increase computational power. The processors are faster than the ones used for both other parallel machines. The advantage of both, the Cray/SGI T3E and the IBM RS/6000 SP, is the faster network. Both commercial systems are limited with respect to extendability. Figure 5.2 shows some technical specifications of the three different parallel machines. Altogether, a Linux cluster is the best way to get a great performance at a low price.

5.2 Parallel Algorithm

To compute the closed streamlines there are two different tasks. First, we have to compute the critical points in the given vector field. The second task is to compute the closed streamlines by determining the topological skeleton. The next two subsections describe these tasks in detail.

5.2.1 Parallel Computing of Critical Points

To parallelize this algorithm we have to compute all the critical points that are present in the vector field, first. Since we only need the data of the cells, i.e. the position of the vertices and the vectors at these vertices, to determine if there exists a critical point inside the cell and where it is located, we can transfer these tasks to the various clients of the cluster. When the clients receive the index of a cell they compute the critical point and return the position and its type, if they have found one, to the server. All tasks are controlled by a scheduler which is a part of the server.

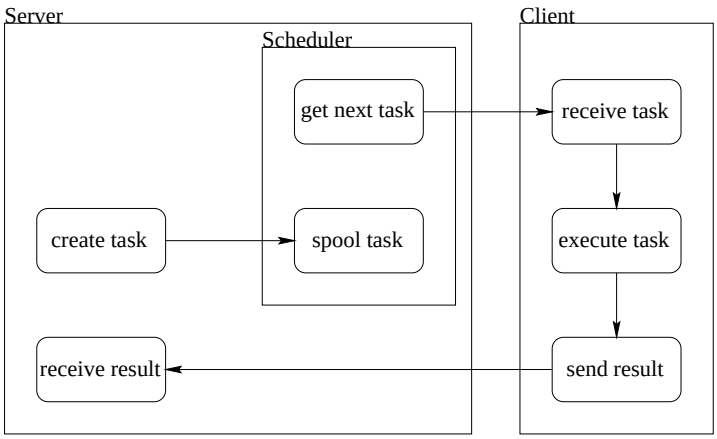


Figure 5.3: Scheduling of the tasks.

The scheduling of the tasks works as follows: the server creates one task for each cell containing the index of this cell and queues it in the scheduler. The scheduler itself checks if there are still tasks left and if there is any client that has finished its task yet. If there is more than one client without an active job, the fastest is chosen. Then the next task is sent to this client. The client receives this task, computes the critical point and sends it, if it has found one, back to the server and tells the scheduler that it has finished its job. Since the amount of data to control the clients and transfer the critical points back to the server is very low, we can fully benefit from the performance of each client.

5.2.2 Determining Closed Streamlines in Parallel

According to the motivation there exist two different approaches for parallelization. We experimented with both approaches to find the best one. There are discussed in detail in the following.

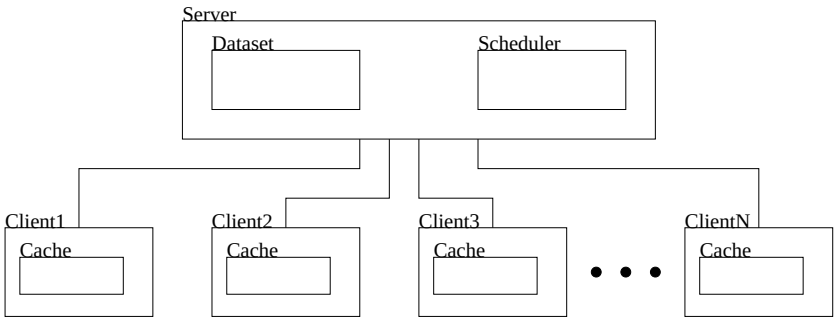


Figure 5.4: Configuration of the parallel program.

5.2.2.1 Subdivision Approach

In our first approach every client got its data from the server. Computing streamlines is a global task since it is not known which region of the vector field this streamline may cross. Therefore it is not possible to simply subdivide the whole dataset into several regions. If we want to subdivide into several regions we have to restart the streamline in another task if it leaves such a region. This usually results in a poor load balancing since it is likely that there are regions that are crossed by only few streamlines. It is possible to switch regions in a particular task. But again, we do not know if a streamline crossed exactly that region we just exchanged.

Therefore we tried a different approach where each request to the dataset on one of the clients results in questioning the server using PVM[GBD+94]. Figure 5.4 shows the configuration. We use caches to avoid asking the server for the same data repeatedly. But due to the slow network connection and the long start-up time for communicating under PVM the transfer of the data took more time than the computation of the streamline even when we used a GigaBit-connection. As a result the parallel version using such a subdivision approach uses more time than the sequential version.

5.2.2.2 Task Driven Approach

After we have computed all critical points, we start streamlines at each saddle point in positive and negative eigendirection with respect to the matrix of the linear interpolant and check for closed streamlines while computing the streamlines as previously described. Computing streamlines is not a local task since the streamlines may cross any region of the flow. Therefore we do not subdivide the data into several blocks like in some rendering tasks [IAO94]. Our implementation uses an approach where we create several tasks each of them representing the whole computation of one streamline starting at a given position. Then we use the scheduler to distribute the tasks to the various clients of our cluster.

Since the data of the vector field including octree and the program fit into 64 MB

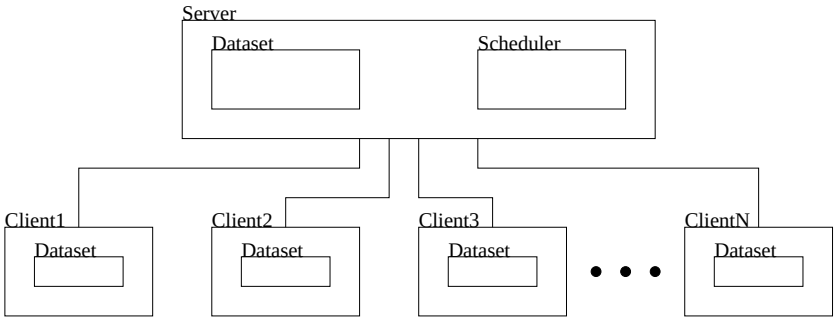


Figure 5.5: Configuration of the parallel program.

of RAM we decided to use a configuration where every client loads the whole dataset into its own memory. This facilitates the fastest possible access to the data. Since the server and every client load the data at the same time there is no time lost because otherwise the clients would simply wait for the server until it has loaded the dataset. When dealing with larger datasets we have to use an out of core method which will be done in the future.

Since we want to spread tasks that represent the whole computation of one streamline, each task contains two items: a point where the streamline has to start and the integration direction. The other data that is needed for the computation is already present at each client because the client has loaded the whole dataset yet. Due to the minimal amount of data of each task the communication cost which is produced by migrating tasks is very low.

To distribute the tasks to the various clients we use the previously described scheduler: the server determines the start positions of the streamline using each saddle point found in the vector field. Then a task containing this start position and the integration direction is created and spooled into the queue of the scheduler, while the scheduler sends the next job to the fastest client that has no active job. The client receives this task, searches for closed streamlines and sends it, if it has found one, back to the server. Again, the amount of data to control the clients and transfer the closed streamlines back to the server is very low, so that we can fully benefit from the performance of each client.

5.3 Results

Our algorithm is implemented in C++, while the server communicates with the clients using PVM[GBD+94]. The different tasks are encapsulated in C++-classes. This facilitates that the tasks can transfer themselves to the client on demand and the clients only need to call a method to execute the received task.

To test the performance of our implementation we mainly use two different systems.

One is a Linux cluster consisting of seven clients. Each node is equipped with an AMD Duron 600 or AMD Duron 700 processor and 64 MB of RAM. The server is a multiprocessor computer with two Pentium III 500 processors. The second system is based on some of our desktop computers with a Pentium II 350. We use Linux and normal PC components since this is a cheap way to get a great performance compared to other parallel computers. In order to get a more heterogeneous configuration we mix both systems by using all Linux computers available in our group for a last performance test.

As a test dataset we use the same simulated dataset as in the previous chapter. The vector field has 362 critical points and for the topology including closed streamlines about six hundred streamlines have to be computed.

Processor	Floating-point index
Pentium II 350	2.404
Pentium III 500	3.561
AMD Athlon 650	5.163
AMD Duron 600	4.768
AMD Duron 700	5.547
Intel Celeron 800	6.125
AMD Thunderbird 1400	11.227

Figure 5.6: Floating-point indices of the different processors.

To determine the optimal timing of our algorithm we used the benchmark utility *nbench*[1] in order to get a suitable ratio between the speeds of the processors. *Nbench* is a port to Linux/Unix of release 2 of BYTE Magazine's BYTEmark benchmark program[2]. We computed the *floating-point index* of each processor which gives the relative speed of the floating-point unit compared to an AMD K6-233 processor. The results can be found in figure 5.6. Using these values we computed the floating-point index of the whole parallel machine by summing up the indices corresponding to the involved processors and calculated the optimal runtime by neglecting the communication cost between server and clients.

Figures 5.7 and 5.8 show the timings on the desktop computers. The cluster consists of up to five machines. The optimal timings are displayed using a dashed line while the real timings are shown by a solid line. This configuration is very suitable for testing the scalability of our implementation because every computer has identical performance. Obviously, the computation time is halved if the number of processors is doubled which indicates a good scalability of our implementation.

[1] http://www.tux.org/~mayer/linux/bmark.html
[2] http://www.byte.com/bmark/bmark.htm

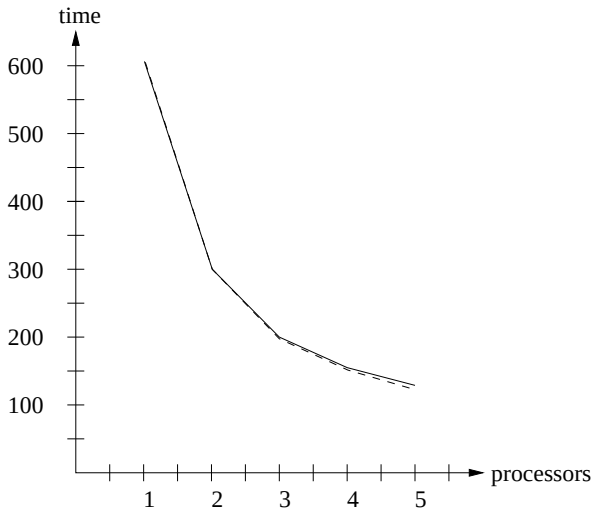


Figure 5.7: Time needed to compute closed streamlines using Pentium PII-350 processors displayed as graph.

# CPUs	Time	Optimum
1	612s	—
2	306s	306s
3	205s	204s
4	158s	153s
5	134s	122s

Figure 5.8: Time needed to compute closed streamlines using Pentium PII-350 processors shown in a table.

The timings of the algorithm running on our Linux cluster with up to seven clients is displayed in figures 5.9 and 5.10. Again, the optimal timings are displayed using a dashed line while the real timings are shown by a solid line. Since the server has two processors there are always running at least two tasks at the same time on this machine. Adding more clients to the Linux cluster the time needed for the algorithm is reduced correspondingly to the speed of its processor. Again, we can see that we nearly benefit from the full performance of each client due to the minimal communication between server and client as can be seen from the difference between the optimal and the real timings.

In our next test we also used the Linux desktop machines in all the offices of our visualization group. This resulted in a parallel machine consisting of six Pentium II-350, two AMD Athlon 650, one dual processor machine with two Pentium III-500, four AMD Duron 600, and three AMD Duron 700. Altogether, the algorithm used seventeen

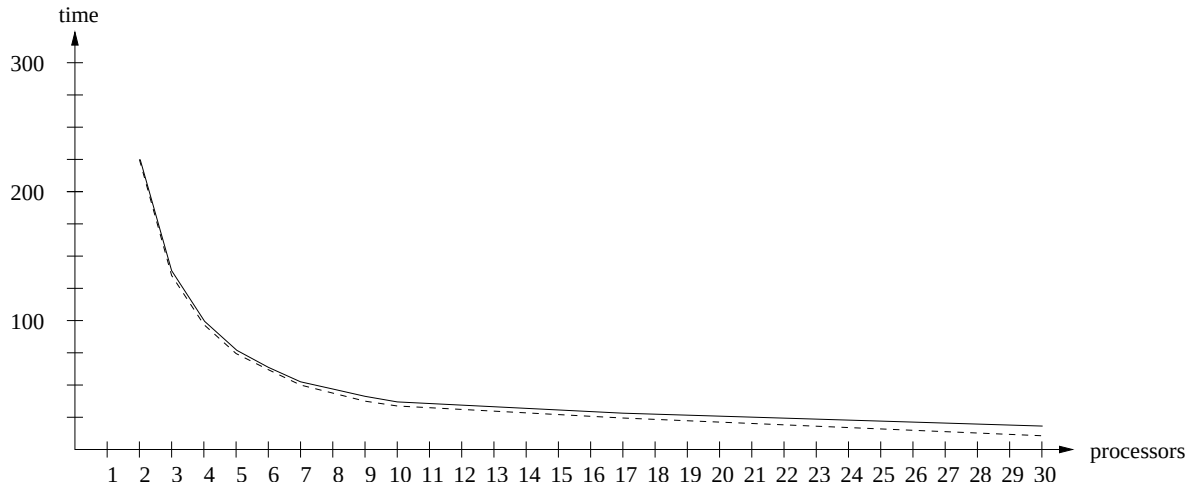


Figure 5.9: Time needed to compute closed streamlines using a Linux cluster with AMD Duron 600 and AMD Duron 700 processors displayed as graph.

# CPUs	Time	Optimum
2	224s	—
3	138s	134s
4	99s	96s
5	77s	74s
6	63s	61s
7	53s	50s
8	46s	43s
9	39s	37s
17	28s	24s
30	17s	9s

Figure 5.10: Time needed to compute closed streamlines using a Linux cluster with AMD Duron 600 and AMD Duron 700 processors shown in a table.

processors and it took 28 seconds to compute all closed streamlines that are present in our test dataset. As expected, this is faster than using the cluster alone corresponding to the speed of the processors and slightly slower than the optimal runtime of 24 seconds. This also tests our implementation in a more heterogeneous parallel machine due to the different speeds of the processors. It shows that we can decrease the time needed for the computation by adding more processors no matter what sort of machine it is.

Then we also added the Linux machines in our student rooms for a last test. These are five machines equipped with an Intel Celeron 800, two machines with a Pentium III-500, and six with an AMD Thunderbird 1400 processor. So we end up with 30

processors. Our algorithm needed 17 seconds. Compared to the optimal timing of 9 seconds this is a little bit too slow. This is due to the slow network connection. Because all computers reside in different areas of our working group and several other processes such as network file system also use this network we do not have the full bandwidth available. Consequently, the communication cost is not neglectable anymore so that the real and the optimal timings differ.

Chapter 6

Closed Streamlines in Time-Dependent Flows

When dealing with closed streamlines one question occurs: how does a closed streamline emerge? Inspired by the books of Abraham and Shaw[AS82] [AS83] [AS84] [AS88] we visualize the evolution of a closed streamline in a planar unsteady flow. We use the third dimension to represent the time. The evolution of a closed streamline can be shown as a tube shaped visulization for the closed streamlines in the various timesteps.

The singularities are used as a starting point for our investigations. Therefore we briefly describe the tracking of the singularities in the next section. This work was done by Tricoche et al. [TSH01]. Then we show how to find and follow a closed streamline over time. In the end we explain the results of our algorithm and explain the limitations of our method.

6.1 Tracking Critical Points

When dealing with time-dependent two-dimensional flows we can use the third dimension to represent time as described in subsection 2.2.4. For tracking the closed streamlines we first determine the behavior of the critical points. For a given cell, the associated interpolant contains, for each value of time t, a single critical point. This is due to the affine linear nature inside the triangles of its restriction to any time plane. Letting the time parameter t move from t_i to t_{i+1}, the critical point position describes a 3D curve. A detailed description of how to find the paths of the critical points can be found in the article of Tricoche et al. [TSH01].

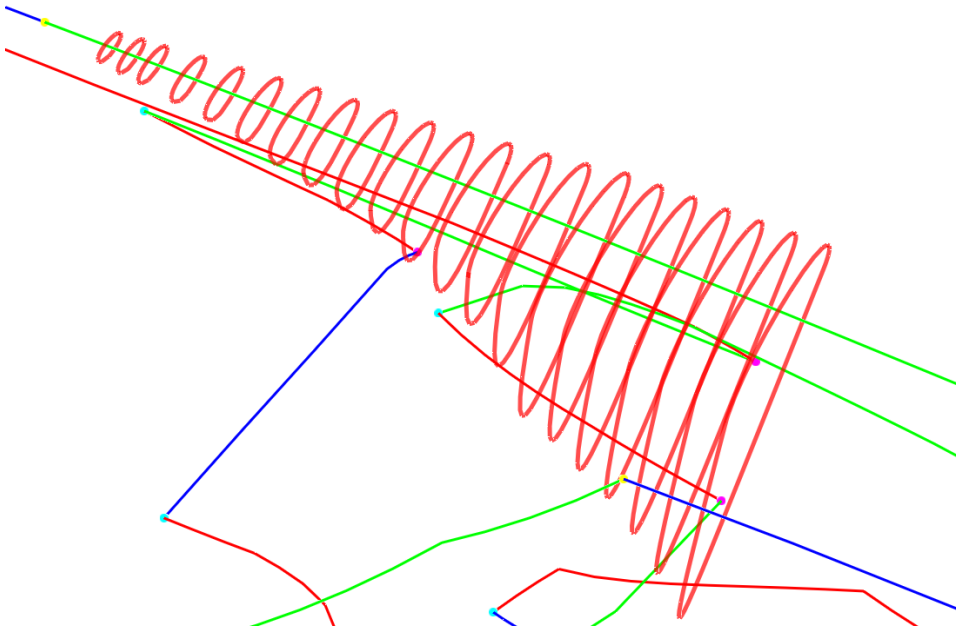


Figure 6.1: Closed streamlines found by the algorithm.

6.2 Following Closed Streamlines

After tracking the singularities, we analyze the vector field in discrete timesteps. There must be a critical point inside each closed streamline. Therefore, we use the critical point path containing a Hopf bifurcation as a starting point for our streamline algorithm from section 4.1 which detects the closed streamline if it exists. We follow the critical point path in discrete steps in positive and negative directions starting at the bifurcation. After we have found the cell cycle containing the closed streamline we find the exact position using the Poincaré-map from section 4.2. As a last test we have to check if the closed streamline really surrounds the critical point. This is necessary because the streamline may have ran into another closed streamline in a totally different region of the flow. Obviously, closed streamlines surrounding the critical point occur only in one of the two temporal directions. We continue by stepping forward in the temporal direction until the closed streamlines reach either another bifurcation which breaks them up or the border of the grid.

Figure 6.1 shows the result of this step. Here we have found the closed streamlines at various timesteps. The closed streamlines are approximated by several line segments. The paths of the critical points are also shown using the same colors as in the original paper [TSH01]. The Hopf bifurcation, where we started to detect the closed streamlines, is marked with a yellow sphere. The different bifurcation types are described in section 2.8. In this example the life cycle of the closed streamline is started by a Hopf

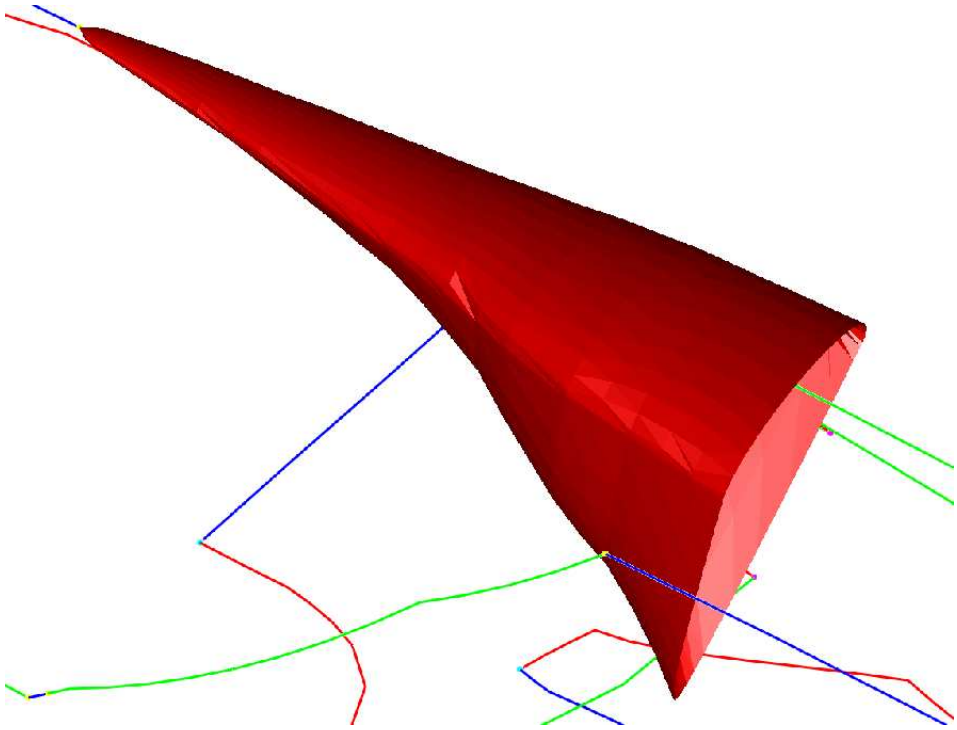


Figure 6.2: Closed streamlines visualized as a tube over time.

bifurcation and terminated by a Periodic Blue Sky in 2D bifurcation.

To visualize the evolution of closed streamlines, we construct tubes from the various closed streamlines similar to the pictures by Abraham and Shaw [AS88]. We construct surfaces consisting of triangles which connect the approximating line segments of the closed streamlines. The bifurcation point is connected to the tube using a parabolic surface approximated with triangles. The result is shown in figure 6.2.

6.3 Results

To test our method, we have created a synthetic vector field containing four critical points. The position of the critical points is a function of time, describing closed curves in the plane. We have sampled this vector field on a triangular point set for several values of the time parameter. The rotation of the critical points (each with a specific frequency) entails many structural changes for the topology. This is very interesting for our purpose since all different types of bifurcations which create closed streamlines are present.

Figure 6.3 shows the result of our algorithm, where the closed streamlines are shown as red tubes. The upper one and the one on the right are started and terminated by Hopf

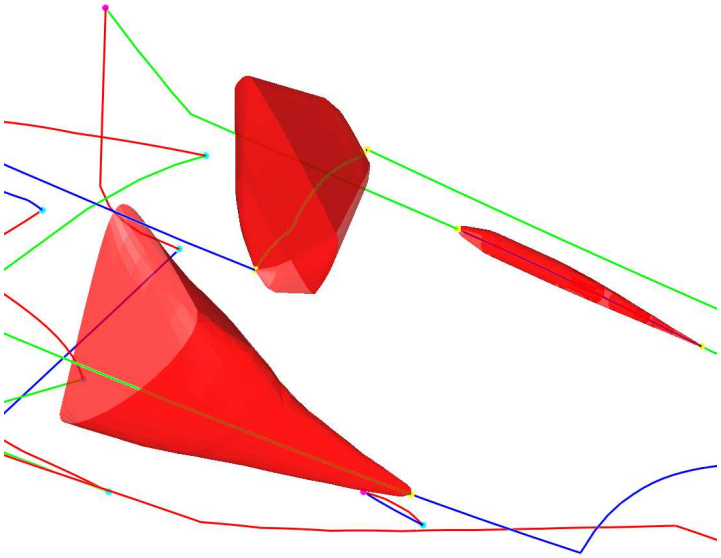


Figure 6.3: Closed streamlines found in a synthetic test dataset.

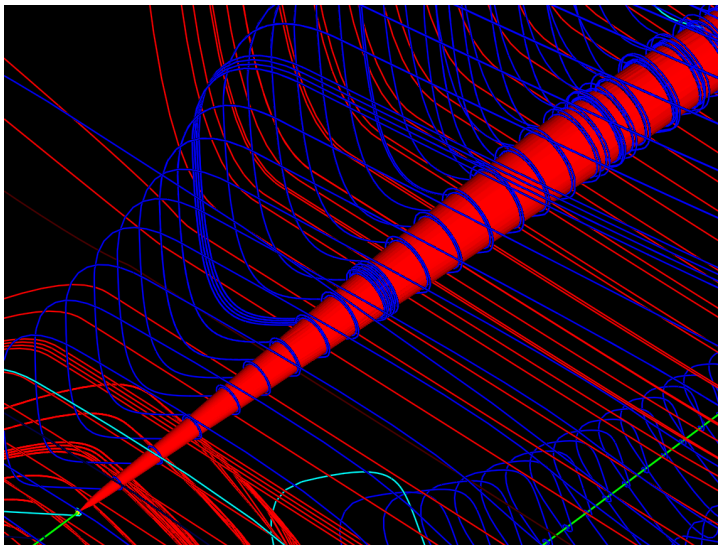


Figure 6.4: Detailed view of a Hopf bifurcation.

bifurcations – shown as a yellow sphere – while the lower closed streamline starts at a Hopf bifurcation and is terminated by a Periodic Blue Sky in 2D bifurcation. Since there is a critical point inside the cell cycle, i.e. the saddle, the flow behaves totally different depending on where a streamline passes the saddle. Therefore the exact localization fails when we are too close to the critical point.

Figures 6.4 and 6.5 show some detailed views of the different bifurcations. Also some

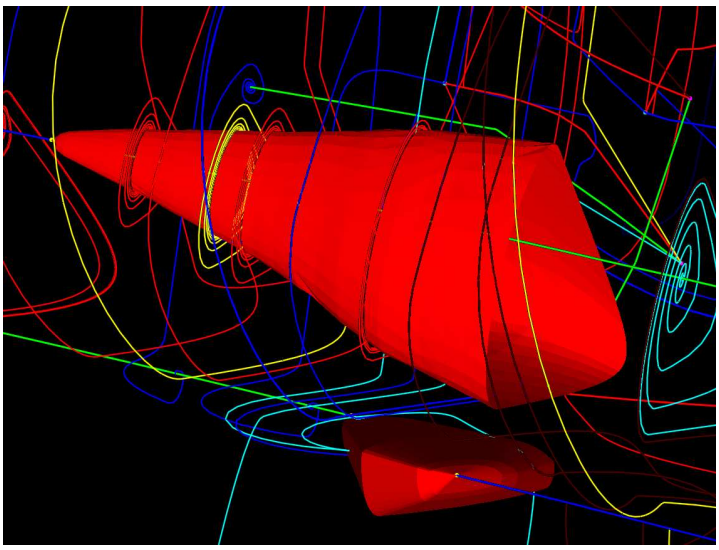


Figure 6.5: Detailed view of a Periodic Blue Sky in 2D bifurcation.

streamlines are drawn to show how these streamlines circle around the limit cycle but never cross it. Figure 6.4 is a closed streamline started by a Hopf bifurcation. The bifurcation is located in the lower left corner. In figure 6.5 the closed streamline is started by a Hopf bifurcation located in the upper left corner. It grows in size until it is terminated by a Periodic Blue Sky in 2D bifurcation. Consequently, the tube visualizing the evolution of the closed streamline does not get closed.

Another dataset we used is a simulation of a swirling jet with an inflow into a steady medium. The simulation uses a cylindrical domain and assumes rotational symmetry, so that we are left with a two-dimensional vector field on a plane through the center axis of the cylinder. In this application one is interested in investigating the turbulence of the vector field and in regions where the fluid stays very long. Swirling jets play a significant role in many combustion processes. It is important to find such recirculation regions indicated by closed instantaneous streamlines. To avoid visual clutter we use only a part of the dataset for our visualization. Figure 6.6 shows the result of our algorithm. The critical point paths are also shown where saddles are colored red, sinks are green, and sources are visualized using blue color. Obviously, in regions where only one saddle point is involved, we cannot find any closed streamline due to the types of bifurcations explained in section 2.8. Most of the closed streamlines emerge at Hopf bifurcations which are marked with a yellow sphere. Therefore, closed streamlines are found where sources and sinks alternate while time propagates, so that we are able to identify the regions where the fluid stays very long.

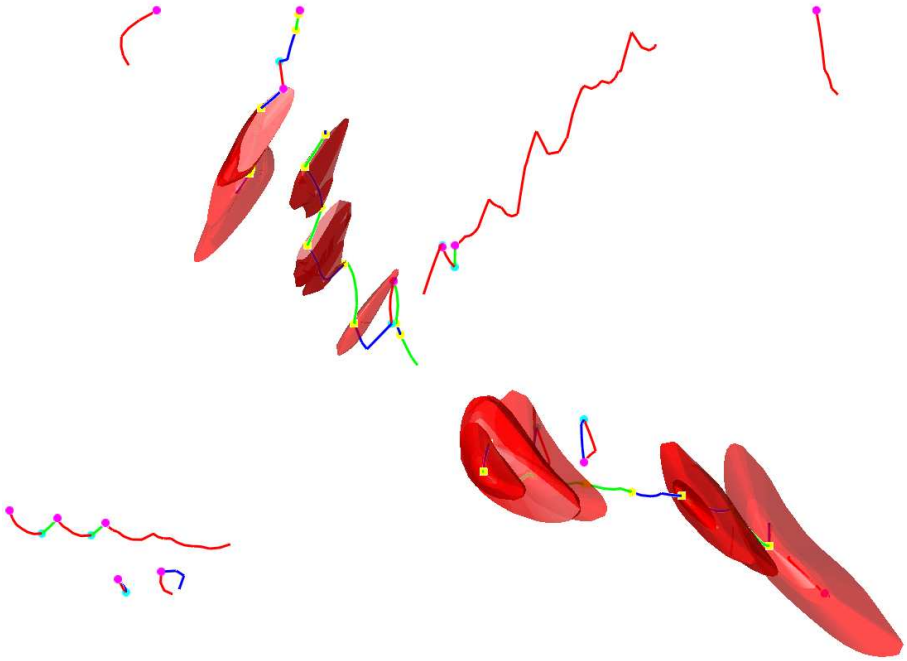


Figure 6.6: Closed streamlines found in a vorticity dataset.

6.4 Limitations

Due to the unstable configuration of the homoclinic connections of the periodic blue sky in 2D bifurcation we actually fail to reach the bifurcation exactly. Our implementation terminates the tube representing the closed streamline slightly too early. Another missing feature in this implementation is to find several closed streamlines around one critical point. This can be accomplished by continuing the integration process. The time slice has to be checked for closed streamlines again near the last limit cycle that was found.

Chapter 7

Closed Streamlines in 3D Vector Fields

Closed streamlines can be found in three dimensional vector fields also. For instance, the *Terrestrial Planet Finder Mission* of NASA deals with stable manifolds where 3D periodic halo orbits play an important role. These orbits are nothing else than closed streamlines in a three dimensional vector field. Figure 7.1 shows an example.

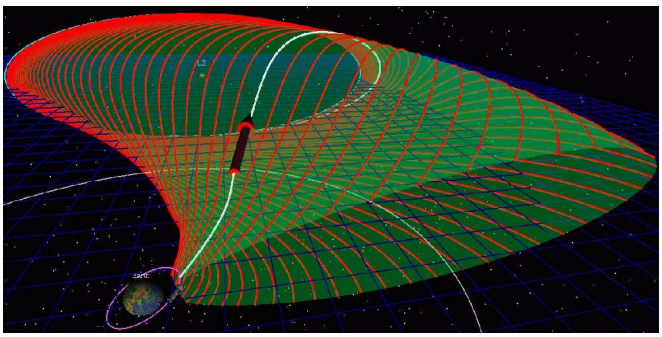


Figure 7.1: Terrestrial Planet Finder Mission (image courtesy of Ken Museth, Caltech[MBL01]).

The next section describes how to detect closed streamlines in three dimensional vector fields. It shows the differences between the two dimensional case both in theory and the algorithm itself. In the end we present the results of the algorithm.

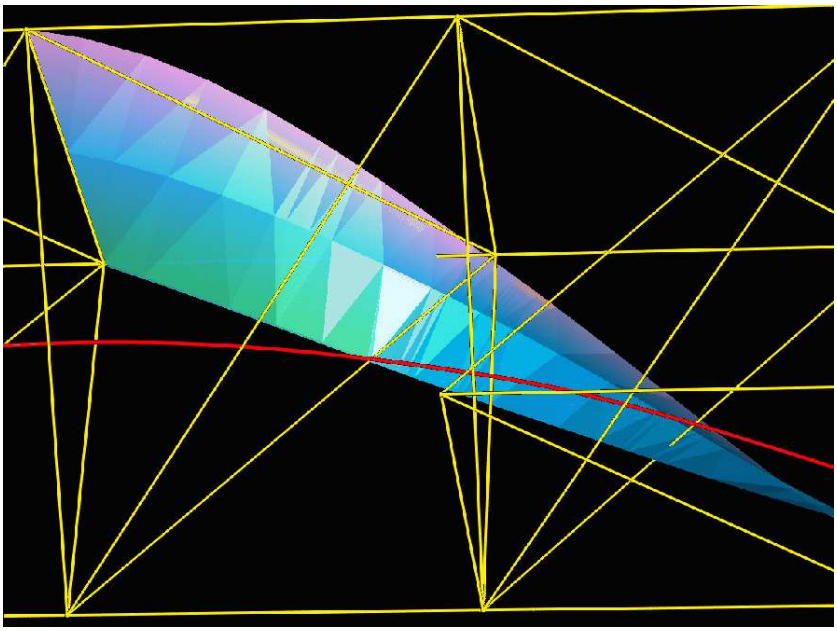


Figure 7.2: Backward integrated surface.

7.1 Detecting Closed Streamlines in 3D Vector Fields

Although the principle to detect closed streamlines in a three dimensional vector field is similar to the two dimensional case there are some differences. We will describe the theoretical and algorithmic differences and similarities in the next two subsections.

7.1.1 Theory

The data is given on a tetrahedral grid. But the principle should work on other cell types as well. The detection of a cell cycle works the same as in definition 4.1.2. Of course, the cells are three dimensional in this case. To check if we can leave the cell cycle we have to consider every backward integrated streamline starting at an arbitrary point on a face of the boundary of the cell cycle. Looking at the edges of a face we can see directly that it is not sufficient to just integrate streamlines backwards. Figure 7.2 shows an example. We integrated a streamsurface backwards starting at an edge of the cell cycle. The streamlines starting at the vertices of that edge leave the cell cycle earlier than the complete surface. So it may be possible that a part of the streamsurface stays inside the cell cycle although the backward integrated streamlines starting at the vertices leave it. Consequently, we have to find another definition for exits.

Definition 7.1.1 (Potential Exit Edges)
Let C be a cell cycle in a given tetrahedral grid G as in Definition 4.1.2. Then we call every edge at the boundary of the cell cycle a ***potential exit edge****. Analogue to the two dimensional case we define a line on a boundary face where the vector field is tangential to the face as a* ***potential exit edge*** *also.*

Due to the fact that we use linear interpolation inside the tetrahedrons we can show that there will be at least a straight line on the face where the vector field is tangential to the face or the whole face is tangential to the vector field.

Theorem 7.1.2
Let F be a triangular face of a tetrahedral. The vectors v_1, v_2, and v_3 are the vector values and p_1, p_2, and p_3 the positions of the vertices of the face F. The vectors inside the tetrahedron are interpolated linearly. Then all vectors that are tangential to the face are on a straight line or all vectors inside the face are tangential.

Proof:
Let v_1, v_2, v_3, p_1, p_2, and p_3 the vectors of respectively positions at the vertices of the face F. Let n be an orthogonal vector to the face F. We can interpolate a vector inside the face F by using the barycentric coordinates:

$$v = \alpha \cdot v_1 + \beta \cdot v_2 + \gamma \cdot v_3$$

Every vector that is tangential to the face F is orthogonal to the vector n. Therefore:

$$n \cdot v = \alpha n \cdot v_1 + \beta n \cdot v_2 + \gamma n \cdot v_3$$

A property of the barycentric coordinates is that they sum up to 1:

$$\alpha + \beta + \gamma = 1$$

So we have two equations and three variables. This leads to an at least one dimensional solution of linear equations if there is any solution. ❑

Remark 7.1.3
Because of theorem 7.1.2 we do not need to consider any isolated point on a face where the vector field is tangential to the face because this cannot occur.

When dealing with edges as exits we have to compute a streamsurface instead of streamlines to consider every point on an exit edge. This leads us to the following notation.

Notation 7.1.4 (Backward integrated streamsurface)
We use the term ***backward integrated streamsurface*** *to describe the streamsurface we integrate by inverting the vectors of the vector field starting at a potential exit edge in order to validate this exit edge.*

Analogue to definition 4.1.5 we define *real exit edges*.

Definition 7.1.5 (Real exit edge)
Let E be a potential exit edge of a given cell cycle C as in definition 7.1.1. If the backward integrated streamsurface does not completely *leave the cell cycle after one full turn through C then this edge is called a* ***real exit edge****.*

For the backward integrated streamsurface we use a simplified version of the streamsurface algorithm introduced by Hultquist [Hul92]. Since we do not need a triangulation of the surface we only have to process the integration step of that algorithm. Initially, we start the backward integration at the vertices of the edge. If the distance between these two backward integrations is greater than a special error limit we start a new backward integration in between. This continues with the two neighboring integration processes until we have created an approximation of the streamsurface that respects the given error limit.

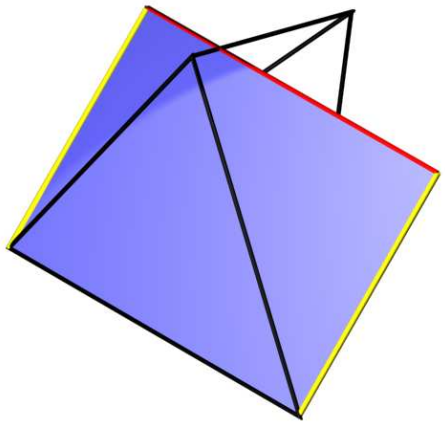


Figure 7.3: Backward integration in one cell.

The integration stops if the whole streamsurface leaves the cell cycle or if we have completed one full turn through the cell cycle. But to construct the surface properly we may have to continue a backward integration process across the boundary of the cell cycle. This is due to the fact that some part of the streamsurface is still inside the cell but the backward integrated streamline has already left it. Figure 7.3 shows a simplified example. Both streamlines at the left and right edge of the surface leave the cell, in fact they leave right after they started. But the integration process must be continued until the whole surface, created inside the cell by these two streamlines, leaves the cell.

With these definitions and motivations we can formulate the main theorem for our algorithm:

Theorem 7.1.6
Let C be a cell cycle as in definition 4.1.2 with no singularity inside and E the set of potential exit edges. If there is no real exit edge among the potential exit edges E or there are no potential exit edges at all then there exists a closed streamline inside the cell cycle.

Proof: *(Sketch)*
Let C be a cell cycle with no real exit edges. Every backward integrated streamsurface leaves the cell cycle C completely. As in the 2D case it is obvious that we cannot leave the cell cycle if every backward integration starting at an arbitrary point on a face of the boundary of the cell cycle C leaves the cell cycle. So we have to prove that the actually integrated streamline cannot leave the cell cycle C.

We look at each face of the boundary of the cell cycle C. Let Q be an arbitrary point on a face F of the boundary of the cell cycle C. Let us assume that the backward integrated streamline starting at Q converges to the actually investigated streamline. We have to show that this is a contradiction.

First case: The edges of face F are exit edges and there is no point on F where the vector field is tangential to F.

From a topological point of view the streamsurfaces starting at all edges of F build a tube and leave the cell cycle. Since the backward integrated streamline starting at Q converges to the actually investigated streamline it does not leave the cell cycle. Consequently, it has to cross the tube built by the streamsurfaces. This contradicts theorem 2.1.8 because streamlines cannot cross each other and therefore a streamline cannot cross a streamsurface.

Second case: There is a potential exit edge e on the face F that is not a part of the boundary of F.

Obviously, the potential exit edge e divides the face F into two parts. In one part there is outflow out of the cell cycle C while at the other part there is inflow into C. We do not need to consider the part with outflow any further because every backward integrated streamline starting at a point of that part immediately leaves the cell cycle C.

The backward integrated surface starting at the potential exit edge e and parts of the backward integrated streamsurfaces starting at the boundary edges of the face F build a tube again from a topological point of view. Consequently, the backward integrated streamline starting at Q has to leave the cell cycle C.

We have shown that the backward integrated streamline starting at the point Q has to leave the cell cycle also. Since there is no backward integrated streamline converging to the actually investigated streamline at all, the streamline will never leave the cell cycle. ❑

7.1.2 Algorithm

With theorem 7.1.6 we are able to describe our algorithm in detail. It is quite similar to the two dimensional case and mainly consists of three different states:

❶ streamline integration: identifying one cell change after the other, check at each cell if we reached a cell cycle.

❷ checking for exits: going backwards through the crossed cells and looking for potential exit edges.

❸ validating exit: integrating backwards a streamsurface from potential exit edges through the whole cell cycle.

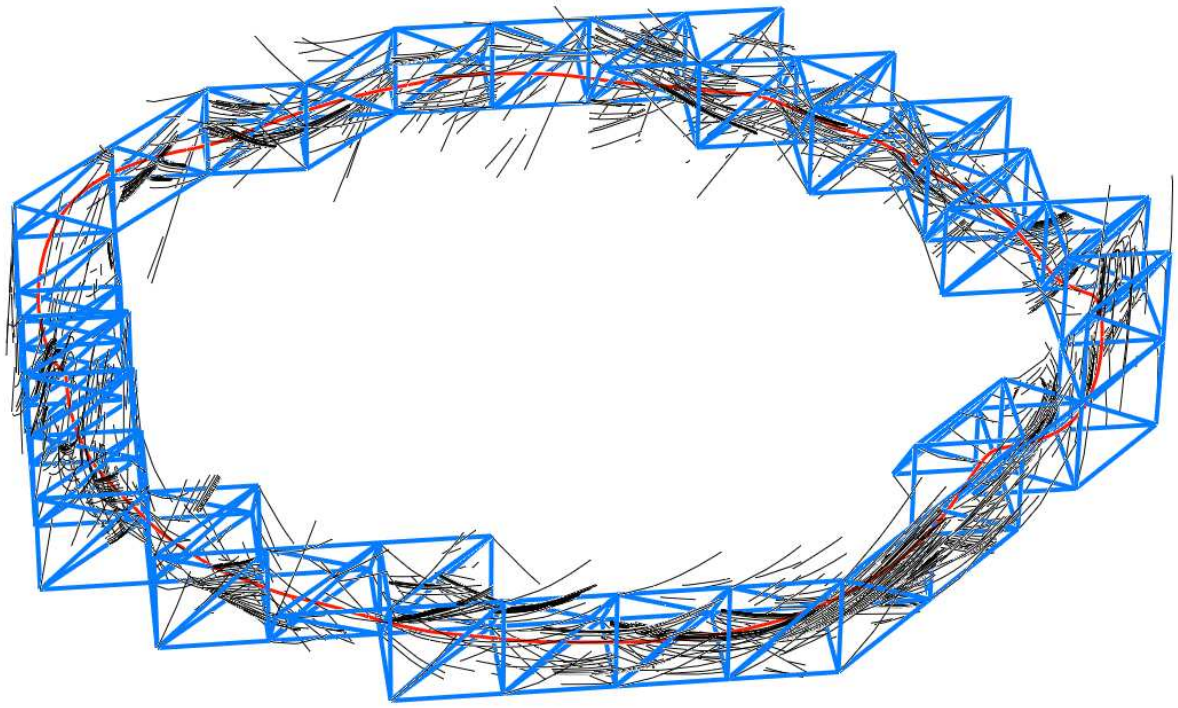


Figure 7.4: Closed streamline including cell cycle and backward integrations.

Figure 7.4 shows an example of our backward integration step. There, also the closed streamline and the cell cycle is shown. Every backward integrated streamsurface leaves the cell cycle. According to theorem 7.1.6, there exists a closed streamline inside this cell cycle. Then we can find the exact location by continuing the integration process of the streamline that we actually investigate until the difference between two successive turns is small enough. This numerical criterion is sufficient in this case since we have shown that the streamline will never leave the cell cycle.

7.2 Results

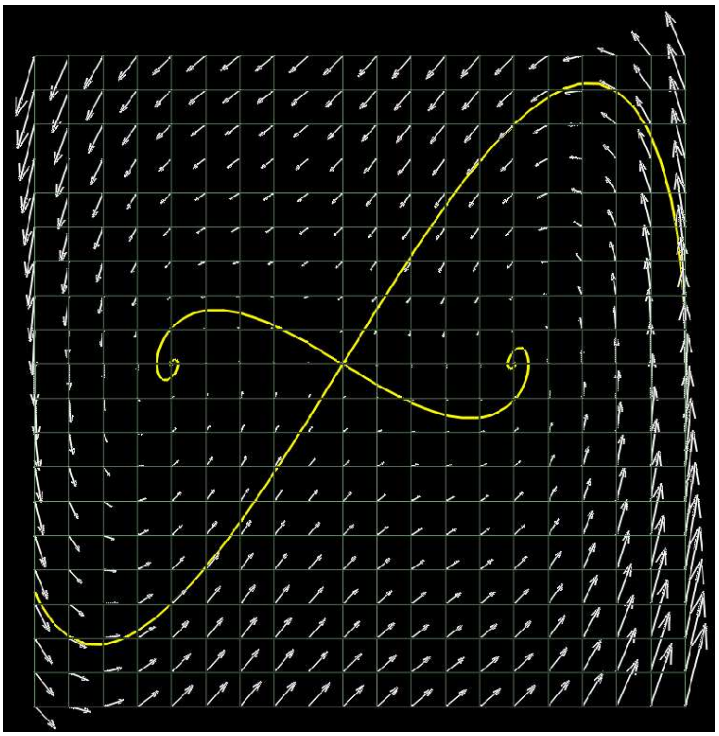


Figure 7.5: Symmetric two dimensional vector field.

To test our implementation we created a synthetic dataset which includes one closed streamline. We first produced a two dimensional vector field. Figure 7.5 shows this vector field. To get an idea of the structure a hedgehog visualization is included. The vector field contains a saddle singularity in the center and two symmetrical sinks. The topological skeleton is shown also. To get a three dimensional flow we rotated the two dimensional vector field around the y-axis. Due to the symmetrical arrangement of the sinks this vector field includes exactly one closed streamline. Figure 7.6 shows the result of our algorithm. To visualize a little bit of the surrounding flow several streamlines are

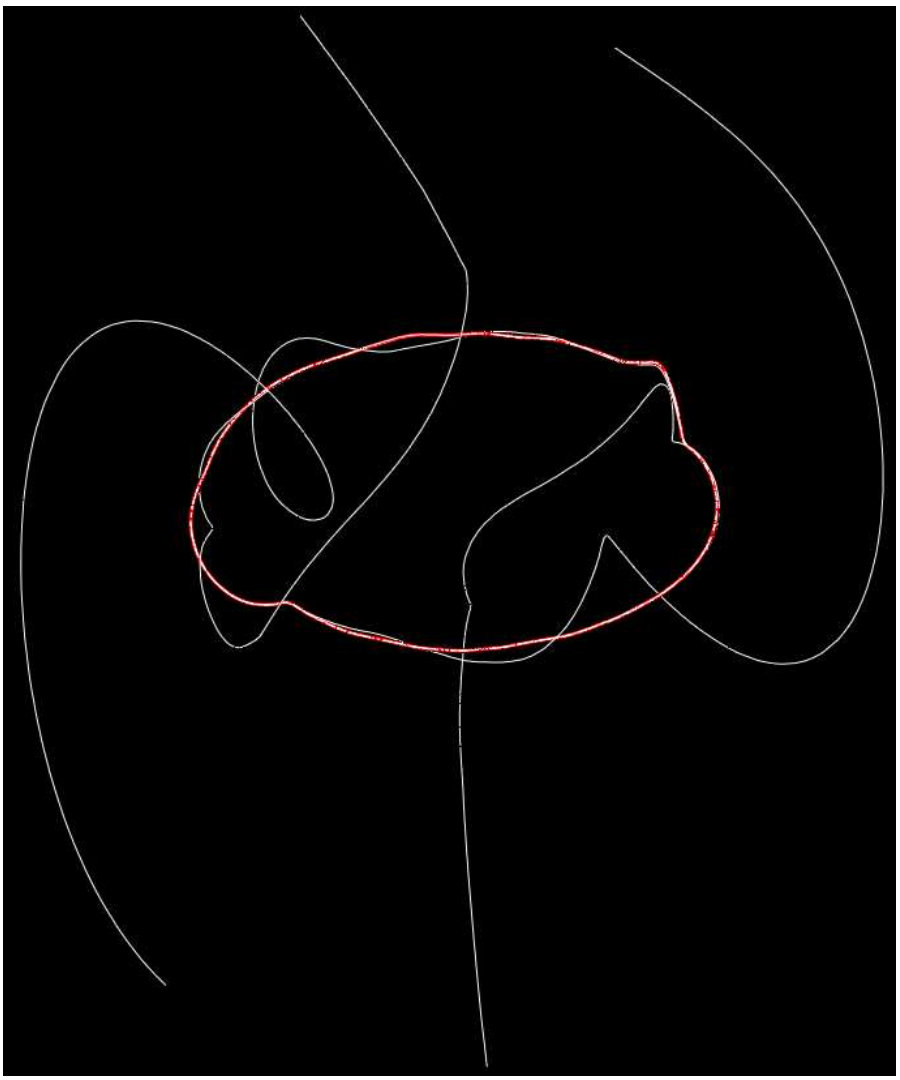


Figure 7.6: Closed streamline in a three dimensional vector field.

drawn. Obviously, every streamline is attracted by the closed streamline. After a short while the streamline spirals around the closed streamline until it completely merges into it. We can see in this example that the closed streamline in this three dimensional flow acts like a sink.

Figure 7.7 shows the same closed streamline with two streamsurfaces. The streamsurfaces are – just like the streamlines – attracted by the closed streamline. The streamsurface gets smaller and smaller while it spirals around the closed streamline. After a few turns around the closed streamline it is only slightly wider then a streamline and finally it totally merges with the closed streamline. We used a rather arbitrary color

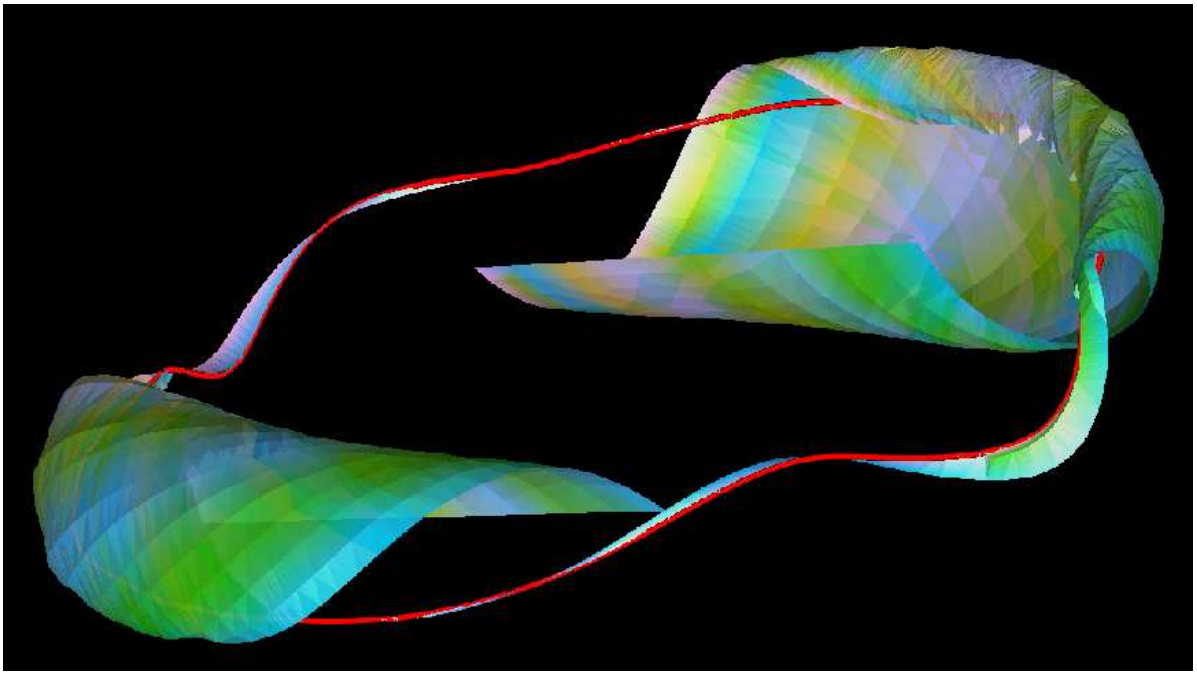


Figure 7.7: Limit cycle in a 3D vector field with streamsurfaces.

scheme for the surface to enhance the three dimensional impression. Both figures 7.6 and 7.7 indicate the potential of this algorithm.

Department of Computer Science, University of Kaiserslautern, Germany

About the Author

Thomas Wischgoll

Dipl.-Inform.

born March 29, 1973 (Borken, Germany)

thomas@wischgoll.tk
http://www.wischgoll.tk

Current Work Address
University of Kaiserslautern
Department of Computer Science
Computer Graphics & CAGD
P.O. Box 3049
D-67653 Kaiserslautern
Germany

Home Address
Phone: ++49 631 340 135 1
Fax: ++49 631 340 135 1

Kanalstraße 67
D-67655 Kaiserslautern
Germany

Research Interests	Computer Graphics, Scientific Visualization, Information Visualization, Virtual Reality, Parallel Computing
Objective	Seeking a research position in a challenging environment where I could put my experience and skills in computer science into practice. Long term goal is a research and teaching position in an educational environment.
Education	B.S. in Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, 1994 M.S. in Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, 1998

	Master Thesis: Visualisierung zeitlicher Distanzen (Visualization of Temporal Distances)
	Ph.D. in Computer Science (expected: December 2002)
Research Related Skills	Programming Languages: C, C++, Java, PASCAL, MODULA-2, LISP, BASIC
	Shell Programming: Bash, Awk, Sed
	Operating Systems: UNIX (Linux, IRIX, HP-UX, AIX, SunOS, Solaris), MS-Windows 95/98/2000/NT
Teaching Related Skills	Lectures: Entwicklung von Software Systemen (Developing Software Systems) *(partly)*
	Training and Support for Students (project related thesis, seminar, practical courses)

Professional Experience

05/95 – 07/97	Student research assistant (Mechanical Engineering), University of Kaiserslautern, Germany
10/97 – 03/98	Student research assistant (Computer Science), University of Kaiserslautern, Germany
10/98 – 01/01	Software development/consulting, ProCAEss, Landau, Germany
10/98 – 10/01	Research/teaching assistant, DFG grant (Computer Science), University of Kaiserslautern, Germany
03/00 – 05/00	Internship (Computer Science), University of California at Davis, California, USA
since 11/01	Research/teaching assistant, DFG grant (Computer Science), University of Kaiserslautern, Germany

Personal Skills	System Hardware, Software Development, Object Oriented Design
	Languages: German, English, Latin
Activities	System administrator for workstation cluster (setup, maintenance, networking, accounting)
	Setup and maintenance of Linux cluster

Responsible for organization of CeBIT Trade Show 2002 exhibit, Hanover, Germany

Publications

Gerik Scheuermann, Thomas Wischgoll, Hans Hagen: Visualization of Temporal Distances, Late Breaking Hot Topics of IEEE Symposium on Visualization 1999, San Francisco, California, 1999, pp. 43–46.

Thomas Wischgoll, Gerik Scheuermann: Detection and Visualization of Planar Closed Streamlines, IEEE Transactions on Visualization and Computer Graphics, June 2001, pp. 165–172.

Thomas Wischgoll, Gerik Scheuermann, Hans Hagen: Parallel Detection of Closed Streamlines in Planar Flows, IASTED International Conference on Visualization, Imaging, and Image Processing, M. H. Hamza (ed.), ACTA Press, Anaheim, 2001, pp. 84–88.

Thomas Wischgoll, Gerik Scheuermann, Hans Hagen: Tracking Closed Streamlines in Time-Dependent Planar Flows, Vision, Modeling, and Visualization 2001, Stuttgart, Germany, 2001, pp. 454-474.

Thomas Wischgoll, Gerik Scheuermann, Hans Hagen: Distributed Computation of Planar Closed Streamlines, IS&T/SPIE's Electronic Imaging Visualization and Data Analysis 2002, San Jose, 2002

Xavier Tricoche, Thomas Wischgoll, Gerik Scheuermann, Hans Hagen: Topology Tracking for the Visualization of Time-Dependent Two-Dimensional Flows, Computer and Graphics, 2002.

Thomas Wischgoll, Gerik Scheuermann: 3D Loop Detection and Visualization in Vector Fields, Visualization and Mathematics 2002, Berlin, Germany, 2002.

Thomas Wischgoll, Gerik Scheuermann: Locating Closed Streamlines in 3D Vector Fields, Joint Eurographics and IEEE TCVG Symposium on Data Visualization 2002, Barcelona, Spain, pp. 227–232.

Department of Computer Science, University of Kaiserslautern, Germany

Bibliography

[Aga89] Anant Agarwal. *Analysis of Cache Performance for Operating Systems and Multiprogramming.* Kluwer Academic Press, Boston/Dordrecht/London, 1989.

[AS82] Ralph H. Abraham and Christopher D. Shaw. *Dynamics – The Geometry of Behaviour: Bifurcation Behaviour Part 1: Periodic Behaviour.* Aerial Press, Inc., Santa Cruz, 1982.

[AS83] Ralph. H. Abraham and Christopher. D. Shaw. *Dynamics – The Geometry of Behaviour: Bifurcation Behaviour Part 2: Chaotic Behaviour.* Aerial Press, Inc., Santa Cruz, 1983.

[AS84] Ralph H. Abraham and Christopher D. Shaw. *Dynamics – The Geometry of Behaviour: Bifurcation Behaviour Part 3: Global Behaviour.* Aerial Press, Inc., Santa Cruz, 1984.

[AS88] Ralph H. Abraham and Christopher D. Shaw. *Dynamics – The Geometry of Behaviour: Bifurcation Behaviour Part 4: Bifurcation Behaviour.* Aerial Press, Inc., Santa Cruz, 1988.

[AT72] T. J. Aprille and T. N. Trick. A computer algorithm to determine the steady-state response of nonlinear oscillators. *IEEE Transactions on Circuit Theory*, CT-19(4), July 1972.

[BDH+94] M. Brill, W. Djatschin, H. Hagen, S. V. Klimenko, and H.-C. Rodrian. Streamball techniques for flow visualization. In *Proceedings Visualization '94*, pages 225–231, October 1994.

[BDJ+99] D. Bürkle, M. Dellnitz, O. Junge, M. Rumpf, and M. Spielberg. Visualizing complicated dynamics. In A. Varshney, C. M. Wittenbrink, and H. Hagen, editors, *IEEE Visualization '99 Late Breaking Hot Topics*, pages 33 – 36, San Francisco, 1999.

[CL93] B. Cabral and L. Leedom. Imaging Vector Fields Using Line Integral Convolution. *Computer Graphics (Proc. o SIGGRAPH)*, pages 263 – 270, 1993.

[CPC90] M. S. Chong, A. E. Perry, and B. J. Cantwell. A General Classification of Three-Dimensional Flow Fields. *Physics of Fluids*, A2(5):765–777, 1990.

[Dal83] U. Dallmann. Topological Structures of Three-Dimensional Flow Separations. Technical Report DFVLR-AVA Bericht Nr. 221-82 A 07, Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt e.V., April 1983.

[DJ99] M. Dellnitz and O. Junge. On the Approximation of Complicated Dynamical Behavior. *SIAM Journal on Numerical Analysis*, 36(2):491 – 515, 1999.

[dLPV96] C. W. de Leeuw, F. H. Post, and R. W. Vaatstra. Visualization of turbulent flow by spot noise. In *Virtual Environments and Scientific Visualization '96*, pages 287–295. Springer, 1996.

[dLvW95] C. W. de Leeuw and J. J. van Wijk. Enhanced spot noise for vector field visualization. In *IEEE Visualization '95 Proceedings*, pages 233–239. IEEE Computer Society, October 1995.

[FC95] Lisa K. Forssel and S. D. Cohen. Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):133–141, 1995.

[FDM+97] Georg Fischel, Helmut Doleisch, Lukas Mroz, Helwig Löffelmann, and Eduard Gröller. Case study: visualizing various properties of dynamical systems. In *Proceedings of the Sixth International Workshop on Digital Image Processing and Computer Graphics (SPIE DIP-97)*, pages 146–154, Vienna, Austria, October 1997.

[Feh69] F. Fehlberg. Klassische Runge-Kutta-Formeln fünfter und siebter Ordnung mit Schrittweiten-Kontrolle. *Computing*, 4, 1969.

[GBD+94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, 1994.

[GH83] J. Guckenheimer and P. Holmes. *Dynamical Systems and Bifurcation of Vector Fields*. Springer, New York, 1983.

[GLL91] A. Globus, C. Levit, and T. Lasinski. A Tool for Visualizing the Topology of Three-Dimensional Vector Fields. In G. M. Nielson and L. Rosenblum, editors, *IEEE Visualization '91*, pages 33 – 40, San Diego, 1991.

[GLW97] Eduard Gröller, Helwig Löffelmann, and Rainer Wegenkittl. Visualization of Analytically Defined Dynamical Systems. In *Proceedings of Dagstuhl '97*, pages 71–82. IEEE Scientific Visualization, 1997.

[Guc00] John Guckenheimer. Numerical analysis of dynamical systems, 2000.

[Hai99] R. Haimes. Using Residence Time for the Extraction of Recirculation Regions. *AIAA Paper 99-3291*, 1999.

[Han93] C. Hansen. Visualization of vector fields (2D and 3D). In *SIGGRAPH '93 Course Notes no. 2, Introduction to Scientific Visualization Tools and Techniques*, August 1993.

[HB90] C. M. Hung and P. G. Buning. Blunt Fin. http://www.nas.nasa.gov/Research/Datasets/Hung/index.shtml, 1990.

[HDER95] D. H. Hepting, G. Derks, D. Edoh, and R. D. Russel. Qualitative analysis of invariant tori in a dynamical system. In G. M. Nielson and D. Silver, editors, *IEEE Visualization '95*, pages 342 – 345, Atlanta, GA, 1995.

[Hel97] J. L. Helman. *Representation and Visualization of Vector Field Topology.* PhD thesis, Stanford University, 1997.

[HH89a] J. L. Helman and L. Hesselink. Automated analysis of fluid flow topology. In *Three-Dimensional Visualization and Display Techniques, SPIE Proceedings Vol. 1083*, pages 144–152, 1989.

[HH89b] J. L. Helman and L. Hesselink. Representation and Display of Vector Field Topology in Fluid Flow Data Sets. *Computer*, 22(8):27–36, 1989.

[HH90] J. L. Helman and L. Hesselink. Surface Representations of Two- and Three-Dimensional Fluid Flow Topology. In G. M. Nielson and B. Shriver, editors, *Visualization in scientific computing*, pages 6–13, Los Alamitos, CA, 1990.

[HH91] J. L. Helman and L. Hesselink. Visualizing Vector Field Topology in Fluid Flows. *IEEE Computer Graphics and Applications*, 11(3):36–46, May 1991.

[HS74] M. W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra.* Academic Press, New York, 1974.

Department of Computer Science, University of Kaiserslautern, Germany

[HS98] Hans-Christian Hege and Detlev Stalling. Fast LIC with piecewise polynomial filter kernels. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization - Algorithms and Applications*, pages 295–314. Springer, 1998.

[Hul92] J. P. M. Hultquist. Constructing Stream Surface in Steady 3D Vector Fields. In *Proceedings IEEE Visualization '92*, pages 171–177. IEEE Computer Society Press, Los Alamitos CA, 1992.

[IAO94] V. Isler, C. Aykanat, and B. Ozguc. Subdivision of 3D space based on the graph partitioning for parallel ray tracing. In P. Brunet and F.W. Jansen, editors, *Photorealistic Rendering in Computer Graphics. Proceedings of the Second Eurographics Workshop on Rendering*, pages 182–90. Springer-Verlag, 1994.

[ID90] A Inselberg and B. Dimsdale. Parallel Coordinates: a Tool for Visualizing Multidimensional Geometry. In *IEEE Visualization '90 Proceedings*, pages 361–378, Los Alamitos, 1990. IEEE Computer Society.

[KB96] Ming-Hoi Kiu and David Banks. Multi-Frequency Noise for LIC. In *Proceedings of IEEE Visualization '96*, pages 121–126, San Francisco, California, October 1996.

[Ken98] D. N. Kenwright. Automatic Detection of Open and Closed Separation and Attachment Lines. In D. Ebert, H. Rushmeier, and H. Hagen, editors, *IEEE Visualization '98*, pages 151–158, Research Triangle Park, NC, 1998.

[Lan93] David A. Lane. Visualization of Time-Dependent Flow Fields. In *IEEE Visualization '93 Proceedings*, pages 32–38, San Jose, California, October 1993.

[Lan94] David A. Lane. UFAT – A Particle Tracer for Time-Dependent Flow Fields. In *Proceedings IEEE Visualization 1994*, pages 257–264. IEEE Computer Society Press, Los Alamitos CA, 1994.

[Löf98] Helwig Löffelmann. *Visualizing Local Properties and Characteristic Structures of Dynamical Systems.* PhD thesis, Technische Universität Wien, 1998.

[LKG97] H. Löffelmann, T. Kučera, and E. Gröller. Visualizing Poincaré Maps Together with the Underlying Flow. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization, Algorithms, Applications, and Numerics*, pages 315–328. Springer, 1997.

[MBL01] Ken Museth, Alan Barr, and Martin W. Lo. Semi-Immersive Space Mission Design and Visualization: Case Study of the "Terrestrial Planet Finder" Mission. In *Proceedings IEEE Visualization 2001*, pages 501–504. IEEE Computer Society Press, Los Alamitos CA, 2001.

[Mul93] Sape Mullender. *Distributed Systems*. ACM Press, New York, 1993.

[NHM97] G. M. Nielson, H. Hagen, and H. Müller, editors. *Scientific Visualization, Overviews, Methodologies, and Techniques*. IEEE Computer Society, Los Alamitos, CA, USA, 1997.

[NJ99] G. M. Nielson and I.-H. Jung. Tools for Computng Tangent Curves for Linearly Varying Vector Fields over Tetrahedral Domains. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):360–372, 1999.

[PC87] A. E. Perry and M. S. Chong. A description of eddying motions and flow patterns using critical point concepts. *Ann. Rev. Fluid Mech.*, pages 127–155, 1987.

[Per84] A. E. Perry. A Study of Non-degenerate Critical Points in Three-Dimensional Flow Fields. Technical Report DFVLR-FB 84-36, Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt e.V., 1984.

[PF74] A. E. Perry and B. D. Fairly. Critical Points in Flow Patterns. *Advances in Geophysics*, 18B:299–315, 1974.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. The University Press, Cambridge, 2nd edition, 1992.

[PvW93] F. H. Post and T. van Walsum. Fluid flow visualization. In H. Hagen, H Müller, and G. M. Nielson, editors, *Focus on Scientific Visualization*, pages 1–40. Springer, 1993.

[RCJ99] Erik Reinhard, Alan Chalmers, and Frederik W. Jansen. Hybrid Scheduling for Parallel Rendering using Coherent Ray Tasks. In *Proceedings of IEEE Parallel Visualization and Graphics Symposium*. ACM SIGGRAPH, New York, 1999.

[Rou98] Robert Roussarie. *Bifurcations of Planar Vector Fields and Hilbert's Sixteenth Problem*. Birkhäuser, Basel, Switzerland, 1998.

[SBH+01] G. Scheuermann, T. Bobach, H. Hagen, K. Mahrous, B. Hahmann, K. I. Joy, and W. Kollmann. A Tetrahedra-Based Stream Surface Algorithm. In *IEEE Visualization '01 Proceedings*, Los Alamitos, 2001. IEEE Computer Society.

[Sch97] Fred B. Schneider. *On concurrent Programming.* Springer, New York, 1997.

[SH95] Detlev Stalling and Hans-Christian Hege. Fast and resolution independent line integral convolution. *Computer Graphics*, 29(Annual Conference Series):249–256, 1995.

[SH96] D. Sujudi and R. Haimes. Integration of Particle and Streamlines in a spatially-decomposed Computation. In *Proceedings of the Parallel Computational Fluid Dynamics.* IEEE Computer Society Press, Los Alamitos CA, 1996.

[SHJK00] G. Scheuermann, B. Hamann, K. I. Joy, and W. Kollmann. Visualizing local Vetor Field Topology. *Journal of Electronic Imaging*, 9(4), 2000.

[SJM96] Han-Wei Shen, Christopher Johnson, and Kwan-Liu Ma. Visualizing Vector Fields using Line Integral Convolution and Dye Advection. In *Proceedings of 1996 Symposium on Volume Visualization*, pages 63–70, 1996.

[SK97] Han-Wei Shen and David Kao. UFLIC: A line integral convolution algorithm for visualizing unsteady flows. In *IEEE Visualization '97 Proceedings.* IEEE Computer Society, October 1997.

[SK98] Han-Wei Shen and David Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):98–108, 1998.

[SVL91] W. J. Schroeder, R. C. Volpe, and W. E. Lorensen. The stream polygon: A technique for 3D vector field visualization. In *IEEE Visualization '91 Proceedings*, pages 123–132. IEEE Computer Society, October 1991.

[SWH99] Gerik Scheuermann, Thomas Wischgoll, and Hans Hagen. Visualization of Temporal Distances. In *Late Breaking Hot Topics of IEEE Symposium on Visualization 1999*, pages 43–46, San Francisco, California, 1999.

[SZH96] Detlev Stalling, M. Zöckler, and Hans-Christian Hege. Parallel Line Integral Convolution. In *Proceedings of the First Eurographics Workshop on Parallel Graphics and Visualization*, pages 111–128, Bristol, U.K., September 1996.

[Tri02] Xavier Tricoche. *Vector and Tensor Field Topology Simplification, Tracking and Visualization.* PhD thesis, University of Kaiserslautern, 2002.

[TSH01] X. Tricoche, G. Scheuermann, and H. Hagen. Topology-Based Visualization of Time-Dependent 2D Vector Fields. In R. Peikert D. Ebert, J. M. Favre, editor, *Proceedings of the Joint Eurographics–IEEE TCVG Symposium on Visualization*, pages 117–126, Ascona, Switzerland, 2001. Springer.

[TWSH02] X. Tricoche, T. Wischgoll, G. Scheuermann, and H. Hagen. Topology Tracking for the Visualization of Time-Dependent Two-Dimensional Flows. *To appear in Computer & Graphics, Special Issue on Data Visualization*, 2002.

[Ung97] T. Ungerer. *Parallelrechner und parallele Programmierung*. Spektrum Akademischer Verlag GmbH, Heidelberg - Berlin, 1997.

[vV87] M. van Veldhuizen. A New Algorithm for the Numerical Approximation of an Invariant Curve. *SIAM Journal on Scientific and Statistical Computing*, 8(6):951 – 962, 1987.

[vW91] J. J. van Wijk. Spot noise: Texture synthesis for data visualization. In *Proceedings SIGGRAPH '91*, pages 309–318, July 1991.

[WFL+00] Pak Chung Wong, Harlan Foote, Ruby Leung, Elizabeth Jurrus, Dan Adams, and Jim Thomas. Vector Fields Simplification – A Case Study of Visualizing Climate Modeling and Simulation Data Sets. In *Proceedings IEEE Visualization 2000*, pages 485–488. IEEE Computer Society Press, Los Alamitos CA, 2000.

[WG97] Rainer Wegenkittl and Eduard Gröller. Fast oriented line integral convolution for vector field visualization via the internet. In *IEEE Visualization '97 Proceedings*, pages 309–316. IEEE Computer Society, October 1997.

[WGP97] Rainer Wegenkittl, Eduard Gröller, and W. Purgathofer. Animation flow fields: Rendering of oriented line integral convolution. In *Computer Animation '97*, pages 15–21. IEEE Computer Society, June 1997.

[WLG97] R. Wegenkittl, H. Löffelmann, and E. Gröller. Visualizing the Behavior of Higher Dimensional Dynamical Systems. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97 Proceedings*, pages 119 – 125, Phoenix, AZ, 1997.

[WS02a] Thomas Wischgoll and Gerik Scheuermann. 3D Loop Detection and Visualization in Vector Fields. In *Visualization and Mathematics 2002*, Berlin, Germany, 2002.

[WS02b] Thomas Wischgoll and Gerik Scheuermann. Locating Closed Streamlines in 3D Vector Fields. In *Joint Eurographics–IEEE TCVG Symposium on Data Visualization 2002*, pages 227–232, Barcelona, Spain, 2002.

[WSH01a] Thomas Wischgoll, Gerik Scheuermann, and Hans Hagen. Parallel Detection of Closed Streamlines in Planar Flows. In M. H. Hamza, editor, *IASTED International Conference on Visualization, Imaging, and Image Processing*, pages 84–88, Anaheim, 2001. ACTA Press.

[WSH01b] Thomas Wischgoll, Gerik Scheuermann, and Hans Hagen. Tracking Closed Streamlines in Time-Dependent Planar Flows. In *Vision, Modeling, and Visualization 2001*, pages 454–474, Stuttgart, Germany, 2001.

[WSH02] Thomas Wischgoll, Gerik Scheuermann, and Hans Hagen. Distributed Computation of Planar Closed Streamlines. In *IS&T/SPIE's Electronic Imaging Visualization and Data Analysis 2002*, San Jose, 2002.

[YqSlLs+86] Ye Yan-qian, Cai Sui-lin, Chen Lan-sun, Huang Ke-cheng, Luo Ding-jun, Ma Zhi-en, Wang Er-nian, Wang Ming-shu, and Yang Xin-an. *Theory of Limit Cycles*. American Mathematical Society, Providence - Rhode Island, 1986.

[ZSH96] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Parallel line integral convolution. In *Parallel Computing*, volume 23, 1996.