AUTOMATIC ILLUSTRATION OF OCEAN CURRENTS

BY

MATTHEW QUINN
BA, Skidmore College, 1999

THESIS

Submitted to the University of New Hampshire
In Partial Fulfillment of
The Requirements for the Degree of

Master of Science
In
Computer Science

September, 2005

This thesis has been examined and approved.

Thesis Director, Colin Ware,
Professor

R. Daniel Bergeron,
Professor

Alejo Hausner,
Assistant Professor

Date

DEDICATION

To Alan Quinn, Anna Quinn and Kate Scheuring for their infinite love and support.

ACKNOWLEDGEMENTS

Foremost, I would like to thank Colin Ware for approaching me with an interesting and exciting research topic and for providing the resources that allowed me to complete this project in a timely fashion. I would also like to thank Colin for his patience and guidance throughout this research; it has been an enlightening and enriching experience for me.

I am grateful to Jamie Pringle for providing us with Finite Volume Coastal Ocean Model output data for the Gulf of Maine region, and for his advice. I would also like to thank Roland Arsenault for his advice and source code that translated data from an irregular grid to a regular grid.

Finally, I would like to thank the members of my thesis committee, R. Daniel Bergeron and Alejo Hausner, for their time and helpful feedback.

Funding for this project was provided by National Science Foundation Information Technology Research grant 0324899 and by National Oceanic and Atmospheric Administration grant NA17OG2285.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

AUTOMATIC ILLUSTRATION OF OCEAN CURRENTS

By

Matthew Quinn

University of New Hampshire, September, 2005

Illustrations of ocean currents are radically different from the typical images automatically generated from ocean model data. Commonly used visualization methods for this data, such as arrow diagrams and Line Integral Convolution, produce dense images with uniform representation over the entire flow field. In contrast, illustrations mainly consist of long, wide arrows or ribbons that highlight only the major patterns in the flow field.

This thesis presents an algorithm to automatically produce images from ocean model data that have the important characteristics of ocean current illustrations. The resulting images are comprised of a set of long, variable-width ribbons overlaid with wide arrows. The color of the arrows and ribbons depict the temperature and salinity levels, respectively, and the length of the arrows is proportional to the velocity of the underlying currents. Secondary, or minor, currents are represented solely as thin arrows.

The vertices of the ribbons are constructed from sets of proximal, parallel running streamlines that are seeded around the longest and strongest streamlines possible in the domain. The resulting structures are called *super-streamlines*. The algorithm described in this thesis was designed to create super-streamlines from two and three-dimensional data sets. Super-streamlines constructed from three-dimensional data may be drawn as variable-width tubes or ribbons. The algorithm is illustrated by applying it to output from a finite volume model of flow patterns in the Gulf of Maine.

INTRODUCTION

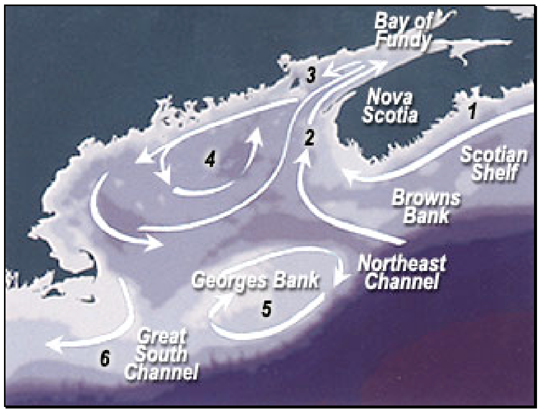


Figure 1: An illustration of currents in the Gulf of Maine.

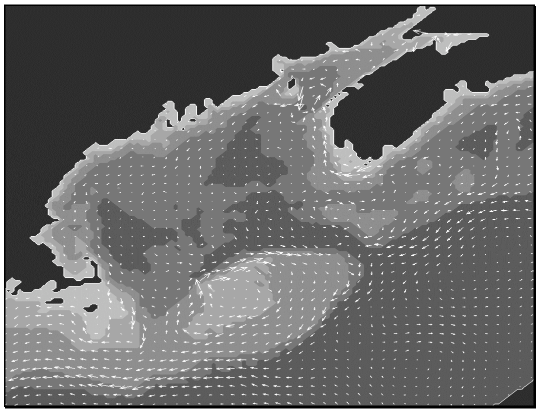


Figure 2: A representation of the Gulf of Maine flow using a regular grid of arrows.

Images depicting the flow of ocean currents are typically produced by one of two methods: through illustrative methods by a person who has expert knowledge of regional flow patterns, or automatically through algorithmic interpretation of ocean model data by a computer graphics system. The resulting images produced by these two methods contrast greatly in their overall appearance and aesthetic. Despite obvious differences, as can be seen in Figures 1 and 2, both methods have merit. Illustrations provide an abstract overview of the behavior of the currents over an area by highlighting only the most important or interesting features. Automatically generated images are produced through precise theoretical methods that may be employed on a variety of data sets with consistent results. Therefore, a method that automatically generates images of ocean currents from ocean model data that have the aesthetic value of illustrative images would benefit from the best qualities of both of these methods.

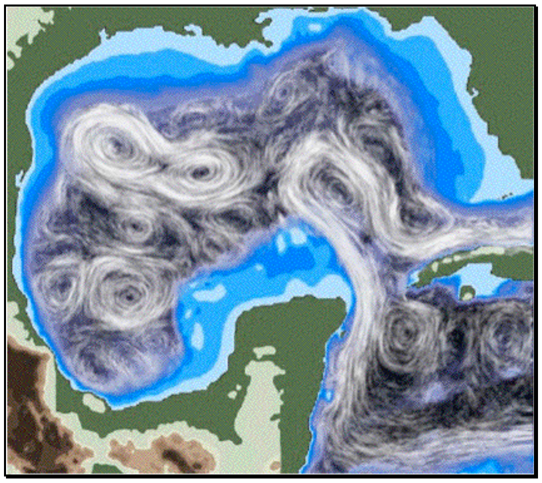


Figure 3: A LIC representation of flows in the Gulf of Mexico

The most commonly used method for automatically generating images of ocean currents is a grid of little arrows over the entire domain, which represent direction and magnitude of the currents in the small local area occupied by the arrow (see Figure 2). The popularity of these diagrams may be attributed to their ease of creation when using scientific computing software packages, such as MatLab®. Despite the prevalence of such images, arrow diagrams are relatively poor at representing global patterns or exposing long-term behavior of a flow field. Additionally, the placement of the arrows may produce aliasing effects, which lead to the perception of false patterns in the data [Ware, 2004].

Line integral convolution (LIC), a dense texture based method used in Figure 3, is also common among automatically generated ocean current images. This method is a popular topic of research in the field of flow visualization due to the highly detailed images produced and the visual beauty of the results. The images produced by LIC are fairly good at exposing global patterns in terms of the flow orientation but lack directional cues and do not inherently show velocity magnitude. In addition, the high level of detail is constant over all areas of the image. This can make it more difficult for one to locate areas where major currents are present.

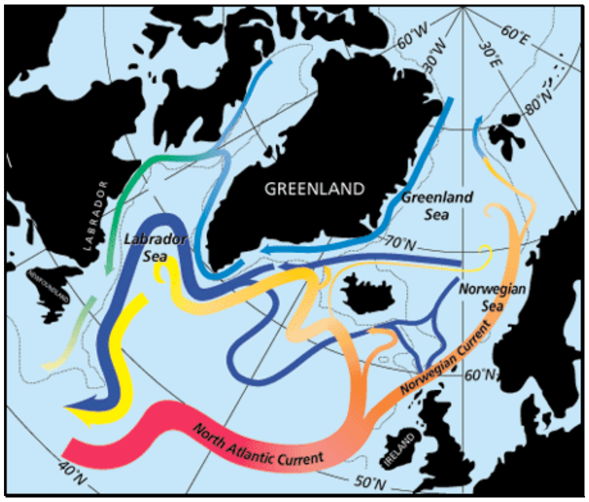
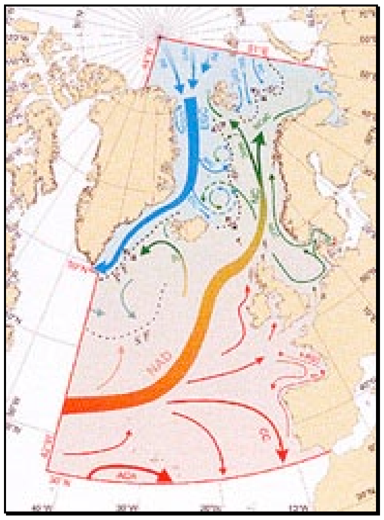


Figure 4: Illustration of North Atlantic currents



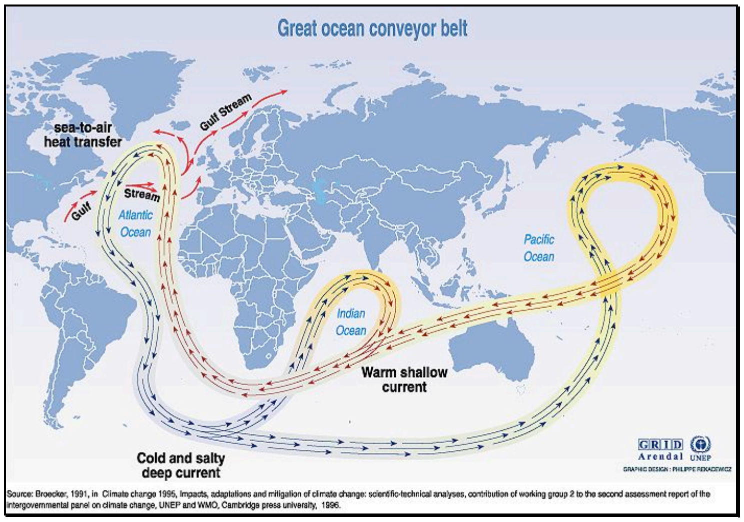

Great ocean conveyor belt
sea-to-air heat transfer
Gulf Stream
Gulf Stream
Atlantic Ocean
Pacific Ocean
Indian Ocean
Warm shallow current
Cold and salty deep current
GRID Arendal
UNEP
GRAPHIC DESIGN : PHILIPPE REKACEWICZ
Source: Broecker, 1991, in Climate change 1995, Impacts, adaptations and mitigation of climate change: scientific-technical analyses, contribution of working group 2 to the second assessment report of the intergovernmental panel on climate change, UNEP and WMO, Cambridge press university, 1996.

Figure 5: Illustration of North Western Europe coastal currents

Figure 6: Illustration of global ocean currents.

Figures 4, 5, and 6 give three examples of hand-designed ocean current illustrations. Although they are each stylistically quite different, they all use smooth broad strokes to depict major flow patterns. Arrow glyphs provide clear directional cues. None of the representations is dense, i.e. all have regions where there is no representation of the flow field. It is common for such diagrams to use the width of the strokes to depict current magnitude as is shown in Figures 4 and 5, although in these examples it is ambiguous as to whether width represents a strong current or a wide current. Another quite common device is to use secondary thin arrows to show secondary flow patterns, as shown with the Gulf Stream in Figure 6. Overall, illustrations have a higher level of abstraction in comparison to automatically generated images, and represent the particular properties of the currents the scientist deems most important.

This thesis presents an algorithm for automatically generating images that have the most important characteristics of common ocean current illustrations showing major current streams. Chapter 1 provides an overview of the field of flow visualization and includes a discussion of research relevant to automatic illustration. Chapter 2 contains information relevant to the ocean model used to illustrate the algorithm. In Chapter 3, the algorithm to automatically generate illustrative ocean current images is described in detail. Conclusions and proposed future research goals are stated in Chapter 4.

CHAPTER 1

FLOW VISUALIZATION

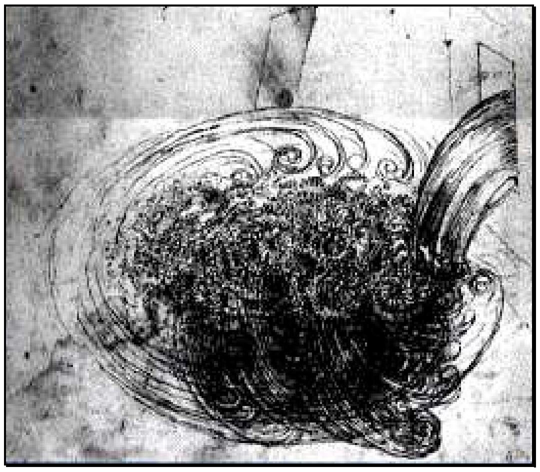


Figure 7: Illustration from a turbulent flow experiment by Leonardo da Vinci

"Flow visualization is the study of methods to display dynamic behavior in liquids and gases" [Ward, 1997]. Traditionally, this has been accomplished by injecting an appropriate visible medium into a volume of transparent material where some flow phenomena are present. The behavior of the visible medium is then recorded to visual media through photography or by human observation and illustration. The earliest such experiments has been attributed to Leonardo da Vinci's illustrations from the mid-fifteenth century of sand particles injected into a box of swirling water [Gad-el-Hak, 2000]. Similar experiments continue today with dye injections used in fluid dynamics and smoke streams used in aerodynamics.

Most modern flow visualization is concerned with computational flow models and the visual representation is done using computer graphics imaging. With the ability to create large, complex computer models and the ever-increasing power and capacity of personal computers, computational fluid dynamics has burgeoned as the dominant area of research in flow visualization. Although the distinction is not always made, flow visualizations of computational fluid dynamic models have been categorized as computational flow visualizations. Since this thesis concerns automatically generating images from ocean model data, from this point forward all references to flow visualization shall refer to computational flow visualizations.

Flow visualization is a broad and growing field within scientific visualization with a wide range of applications, including design for vehicle aerodynamics, engines, propulsion and wind turbines, as well as applications in cardiology, meteorology and, of course, oceanography. Properties of the modeled data used in each of these applications are most often suited to best serve application related tasks. Thus, the models used in flow visualization can vary greatly in many aspects, such as physical shape, dimensionality (2D, 3D, time-varying) and geometry of the data sample point locations. This section presents a generalized overview and categorization of flow visualization methods. A discussion of specific works related to automatic illustration follows.

1.1 <u>Overview of Flow Visualization</u>

In a report on the state of the art of flow visualization, Hauser, Laramee and Doleisch presented a categorization of flow visualization research [Hauser et al., 2002]. Their work classifies the field into four categories: direct flow visualization, flow visualization with integral objects, dense integration-based flow visualization, and flow visualization based on derived data. The following sections provide an overview of flow visualization following the organization presented by Hauser, Laramee and Doleisch. An emphasis is placed on the section concerning flow visualization with integral objects, because topics from this section are most relevant to this thesis.

1.1.1 <u>Direct Flow Visualization</u>

Direct flow visualization methods involve minimal translation and computation between the acquisition of data and rendering. The images produced are intended to be straightforward representations of the data and are often used to verify the data acquisition process. These representations are, however, inferior in portraying the long-term behavior of a flow field.

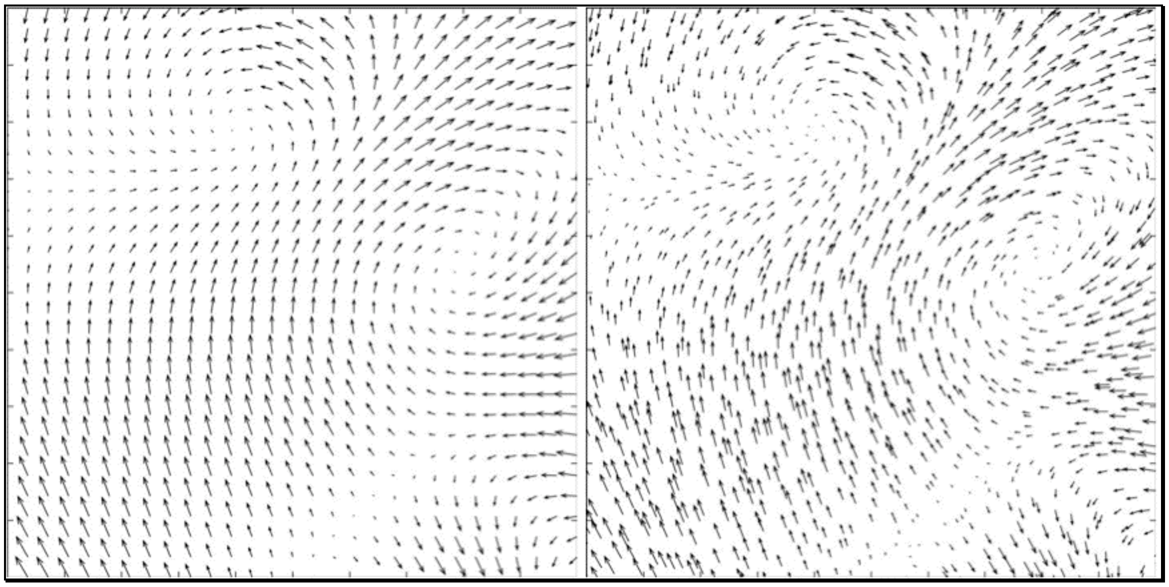


Figure 8: Left- arrow diagram with arrows placed at sample point locations of a regular Cartesian grid. Right – arrow diagram of the same flow pattern with arrows placed on jittered grid.

Most direct representations consist of arrow diagrams and/or contour maps. In arrow diagrams, also called hedgehog visualizations, each arrow directly represents the direction and magnitude of the vector where it is positioned. If the arrows are positioned exactly where the sample points in the data are located, the images may have misleading visual artifacts [Turk & Banks, 1996]. As shown in Figure 8, jittering the arrow locations is a common solution to reduce this effect.

Contour maps in flow visualizations represent one attribute of the flow, typically magnitude or direction, by using a color sequence to depict a range of values over an area. This representation is most often useful for exploring cross sections of three-dimensional data as shown in Figure 9.

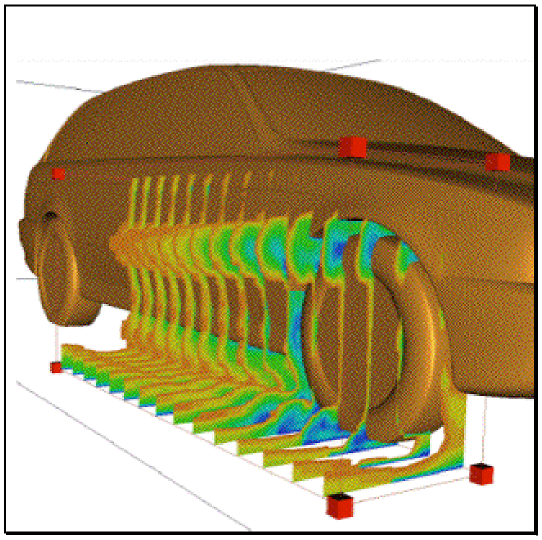


Figure 9: Color-coded contour map cross-sections of flow along an automobile.

Although often useful for initially exploring a data set, direct representations are unfit for exposing larger patterns in the behavior of the flow field. Methods that aim to expose larger patterns or more long-term behavior require task specialized computations prior to rendering.

1.1.2 Flow Visualization With Integral Objects

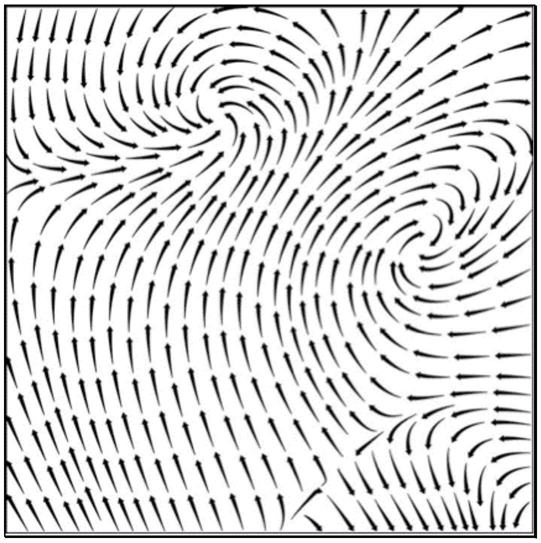


Figure 10: Directional glyphs arranged end to end along evenly-spaced streamlines

In understanding flow it is often useful to determine the trajectories of a set of particles placed into the flow field. The approximated path of a particle is computed using numerical integration. Each point along the path is determined from the position of the previous point and the direction and magnitude of the flow field local to the previous point. The points of a single trajectory or related sets of trajectories are visualized by a curving line or a series of glyphs collectively called *integral objects*.

Many different methods have been developed for approximating integration using discrete values. The simplest method simply multiplies the vector value at the starting point by the time step and sums it with the starting point to find the next location. This can be represented mathematically as,

$$p1 = p0 + v0 * \Delta t$$

where $p0$ is the start point, $v0$ is the velocity vector at that point and Δt is the time step. However, this method, called *Euler's method*, is inaccurate for larger time steps and is often superseded by alternative methods. Commonly, the more accurate, but less efficient *second order Runge-Kutta* technique is used. This method performs the Eulerian method twice: first, to calculate the position at one time step from the starting position, and then again using the average of the vector at the start position and the vector at the position calculated in the first step. This may be represented mathematically as,

$$p1 = p0 + (v0+v1)*\Delta t/2$$

where *v1* is the velocity vector at the position, calculated through Euler's method, that is one time step from *p0* [Schroeder et al., 2004]. A series of points resulting from multiple steps of integration are typically visualized by means of *pathlines*, *streamlines*, *streaklines*, or *timelines*. Each of these integral objects is defined in detail in the following sections.

1.1.2.1 **Pathlines**

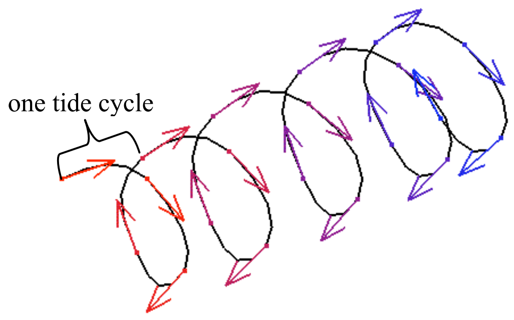


Figure 11: A pathline (black) with arrows indicating direction of travel and progression of time (from red to blue)

Pathlines, also called particle traces, are an approximation of the trajectory of a massless, frictionless particle through time and space from a given start position and time. The control points along the line are determined by repeatedly calculating the position of the particle at the next time step from the position of the particle and the vector at that position for the current time step. Each point on the line represents the particle's location as a function of time. Thus, the construction of a pathline requires vectors that are arranged in multiple time steps.

Although pathlines are useful in many contexts, pathlines may intersect with themselves or other pathlines creating difficulty in determining the behavior of the field, particularly in areas of high turbulence. Figure 11 demonstrates the spiral path common in the Gulf of Maine that is created as the tides move in and out in cycles.

1.1.2.2 <u>Streaklines</u>

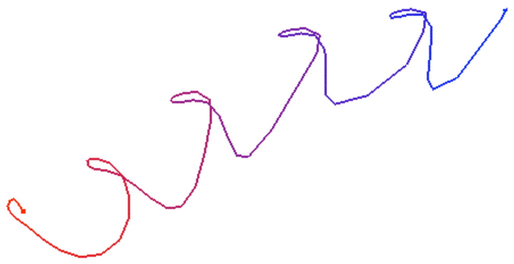


Figure 12: A streakline. All particles along the line originate from the red dot and change in color from red to blue as time increases from time of release.

A streakline is the line through a series of particles at the same instant in time that all have passed through or originated from a common point. The points along the line are connected in the order that they passed through the common point i.e., in ascending order of the time they have "aged" since passing through the common point. For every point on the streakline there is a pathline that passes through it and the point of origin. In a physical model streaklines could be compared to a continuous injection of a smoke stream into a flow field at a constant position. As with pathlines, streaklines require time series data and may produce patterns that are visually complex in turbulent areas.

1.1.2.3 <u>Timelines</u>

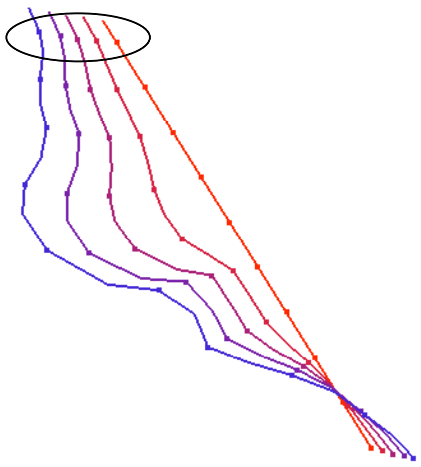


Figure 13: Timelines originating from red line and progressing over time to the blue line. The points enclosed in the circle occur on a common pathline.

A timeline joins the positions of particles released at the same instant in time from an array of insertion points initially arranged in a line orthogonal to the flow direction. For every point on a timeline there exists a point on each previous and subsequent timeline that are a part of a common pathline. For example, the points circled in Figure 13 exist on a common pathline. Timelines advance like wave fronts as time increases from the initial position. Although timelines can produce very effective images conveying flow patterns over large areas and over time, the position and orientation of the initial line of seed points must be determined interactively and these factors largely govern the overall effectiveness of the resulting image.

1.1.2.4 <u>Streamlines</u>

A streamline is an instantaneous path that is everywhere tangential to the vector field. Unlike the previously mentioned objects, streamlines show only instantaneous behavior, thus are independent of time. So, in integration, the value previously described as a time step is an arbitrary constant when used with streamlines. In areas of steady flow i.e., areas the flow exhibits negligible change over time, pathlines and streamlines are identical.

Streamlines may create loops or spirals, but do not cross over themselves or other streamlines seeded at the same instance. Because of this property, streamline seeding algorithms have been developed to effectively display flow patterns by densely seeding streamlines over the entire domain of a flow field. Two of these seeding strategies are discussed in later sections of this chapter.

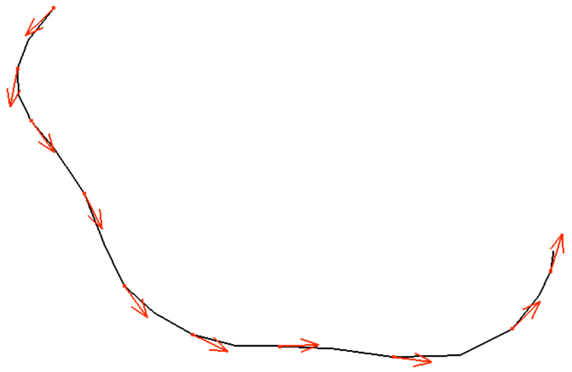


Figure 14: A streamline (black) tangent to all vectors in one instant in time.

1.1.2.5 <u>Stream Surfaces</u>

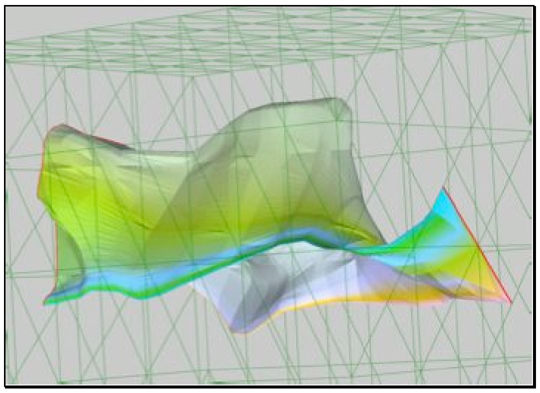


Figure 15: A stream surface originating from red line.

A *stream surface* is a curved surface that is everywhere tangential to the flow field at a given instant that emanates from a line of seed points called a *rake*. A stream surface may alternatively be described as the surface created by an infinite number of streamlines seeded along a common line. Stream surfaces are most often used for interactively exploring dynamic behavior in three-dimensional space.

Hultquist presented a method for approximating and rendering a stream surface as a triangular mesh constructed from points on adjacent streamlines. This was accomplished by expressing the stream surface as a two-dimensional parametric surface embedded in a three-dimensional domain and using a greedy minimal-width tiling strategy to create the mesh [Hultquist, 1992].

van Wijk presented an alternative method for constructing stream surfaces by expressing them as implicit surfaces defined by constant values of a stream function [van Wijk, 1993]. Once a stream function has been calculated, a stream surface may be rendered at the appropriate isovalue using standard volume rendering techniques, such as marching cubes [Lorensen & Cline, 1987].

1.1.3 <u>Dense, Integration-based Flow Visualization</u>

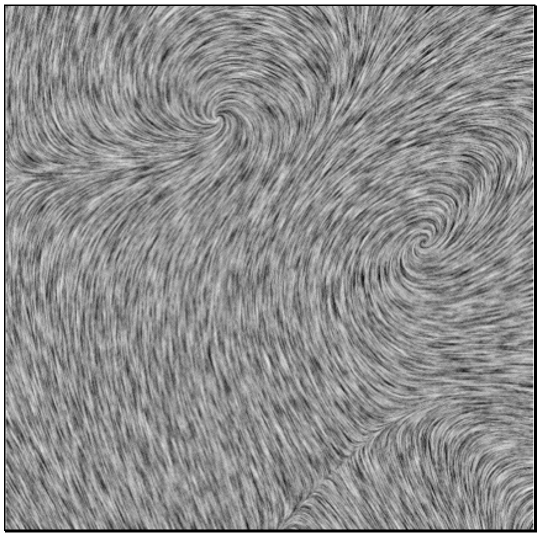


Figure 16: An example of Line Integral Convolution

Dense, integration-based flow visualizations consist of images that have equally dense representations over the entire flow field. These images are typically produced from densely seeded integral objects followed by a specialized rendering process. Spot noise, presented by van Wijk, was the first method to produce densely textured representations of vector fields [van Wijk, 1991]. The spot noise method distributes a series of spots at random locations about the flow domain and then distorts the spots in relation to the underlying vector field.

Line integral convolution, initially introduced by Cabral and Leedom, is a popular method for producing dense representations. In this process, shown in Figure 16, a white noise texture is smoothed along densely seeded streamlines to produce a highly detailed image of the flow field [Cabral & Leedom, 1993].

1.1.4 Flow Visualization Based on Derived Data

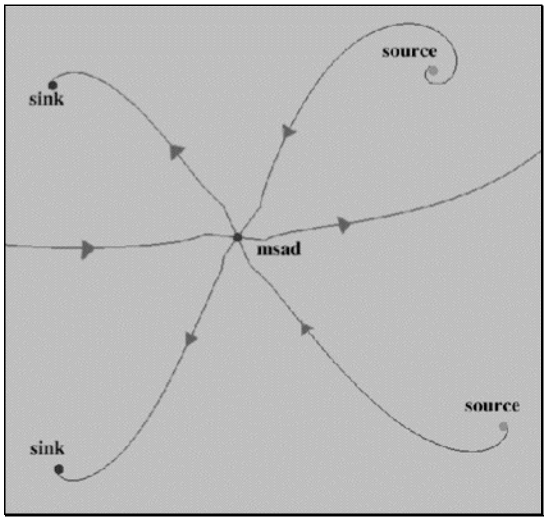


Figure 17: An example of a flow topology visualization

Flow visualizations based on derived data involve additional calculations to acquire new data based on the input data before the rendering process. The properties of the derived data depend on the application. A common application is to reveal flow topology by calculating the location and type of *critical points* or *singularities* in the flow field where the velocity reaches zero. Figure 17 shows a representation of flow topology visualization [Scheuermann et al., 1997].

1.2 Visualization Research Motivated by Illustrative Techniques

Illustrations have a long history in effectively communicating information, from early cave paintings, to the anatomical studies and mechanical designs of Leonardo da Vinci, to their modern use in educational texts. Illustrators have developed a vast number of specialized drawing techniques aimed at efficacious conveyance of concepts and information. These techniques range from use of colors, to individual stroke properties, to the use of abstraction and exaggeration to draw attention to an area.

Because scientific visualizations and illustrations serve a similar objective, i.e., to convey information effectively, some visualization research has drawn from illustrative techniques to improve the overall clarity and style of their images. The following subsections discuss some notable visualization research that was motivated by illustrative techniques.

1.2.1 <u>Computer Generated Pen-and-Ink Illustrations</u>

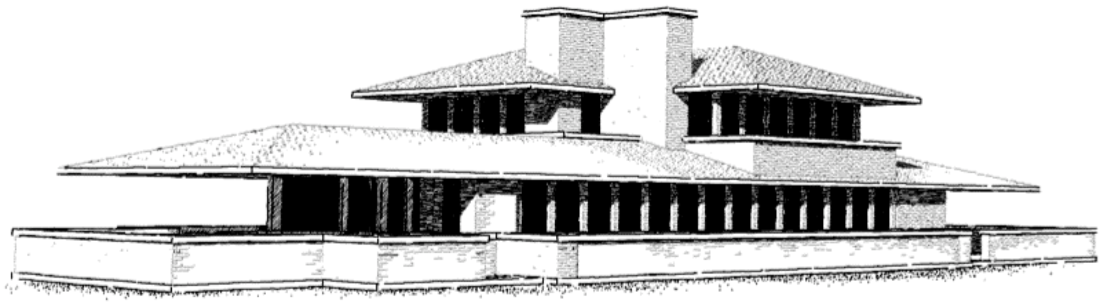


Figure 18: An automatic illustration of the Frank Lloyd Wright "Robie House"

Winkenbach and Salesin introduced a method for simulating pen-and-ink illustrations as used in architectural renderings by implementing a modified graphics pipeline that depicts textures and tones through simulated pen strokes. Individual strokes are controlled by a waviness function and a pressure function, which dictate the distortion of the path and the thickness, respectively. Collections of strokes are organized into a series of stroke textures that are prioritized in a manner that ensures the appropriate density of strokes to properly display the tone, notwithstanding the resolution of the image [Winkenbach & Salesin, 1994].

1.2.2 Illustration-Based Lighting Model

Gooch, Gooch, Shirley and Cohen presented a shading algorithm based on the techniques used in technical illustration. This algorithm is designed to replace the traditional photorealistic Phong lighting model in order to improve the communication of geometric shape and material information. This is accomplished through using luminance and changes in hue from "cool-to-warm" colors to indicate the orientation of the surface, black edge lines to highlight surface boundaries and silhouettes, and white highlights from a single light source [Gooch et al., 1998].

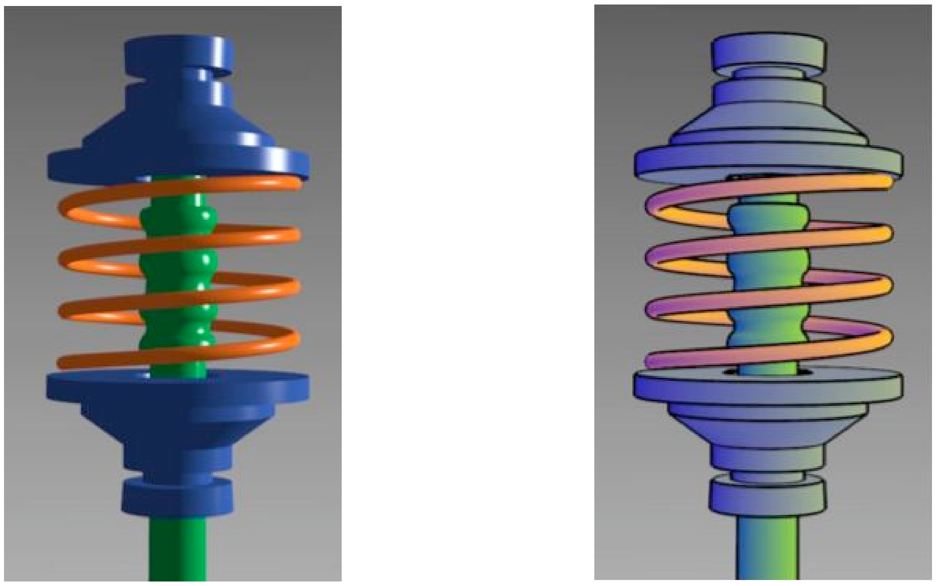


Figure 19: Left – Phong model for a colored object. Right – illustration based model

1.2.3 Volume Rendering Based on Stippling Illustrations

Lu, Morris, Ebert, Rheingans and Hansen presented a volume rendering technique emulating the stippling illustration style. This is accomplished by first generating stipple points based on volume characteristics, such as gradients and resolution, and then calculating points to enhance features at boundaries and silhouettes. The authors note that this method has proven effective in both medical education as well as preliminary data exploration [Lu et al., 2002].

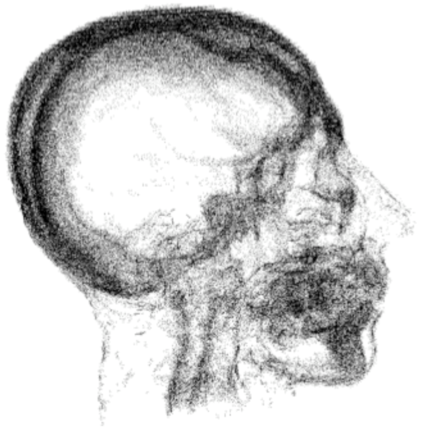


Figure 20: Head volume using automatic stippling illustration

1.3 Flow Visualization Research Related to Automatic Illustrations

Although we are not aware of any prior research that is directly concerned with the problem of representing ocean current flows in a manner similar to hand-drawn arrow diagrams, there have been several prior studies in flow visualization that have been motivated by the work of artists and illustrators.

1.3.1 Visualizing Multi-valued Flow Data Using Concepts from Painting

Kirby, Marmanis and Laidlaw presented a method for representing multiple data values simultaneously following techniques used by artists in oil on canvas paintings [Kirby et al., 1999]. This was accomplished by assigning different variables various graphic attributes, such as: color, opacity and brush-like textured strokes, and then rendering the values in layers. This follows how artists prime a canvas and then apply paint in layers of textured strokes. The goal of this work was to show the values of several distinguishable variables in a manner that exposes their relationship.

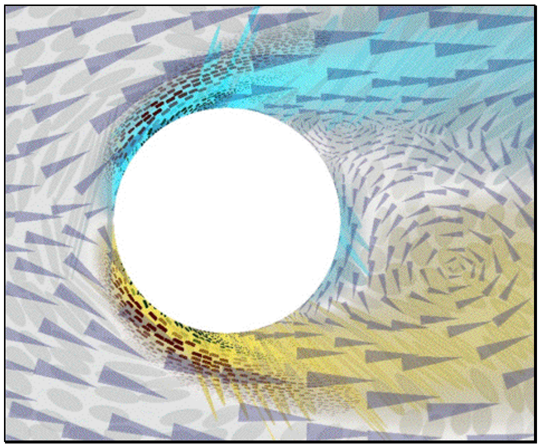


Figure 21: A visualization of flow around a cylinder using concepts from painting. Nine separate values are represented.

The visualization produced by Kirby et al., shown in Figure 21, represents a set of velocity vectors taken *directly* from the data and a set of values *derived* (through calculations) from the data, such as vorticity, rate of strain, turbulent charge and turbulent current. Visualizing the data in this manner allows one to explore the relationships that are underlying the dynamics and kinematics of fluid flow. Additionally, the authors note that the level of detail perceived from the image can be altered with the perspective of the viewer; i.e., the visualization may be viewed at different distances to observe the data at different levels of abstraction. Contrary to our research goals, these images have dense representations throughout the flow field and do not expose major channels of flow.

1.3.2 Image Guided Streamline Placement

Turk and Banks introduced an algorithm for controlling the placement of streamlines in two dimensions using a low-pass-filtered version of the image to guide the process [Turk & Banks, 1996]. The motivation of this work was to automatically produce streamline representations similar to hand-designed illustrations as shown in the top image of Figure 22.

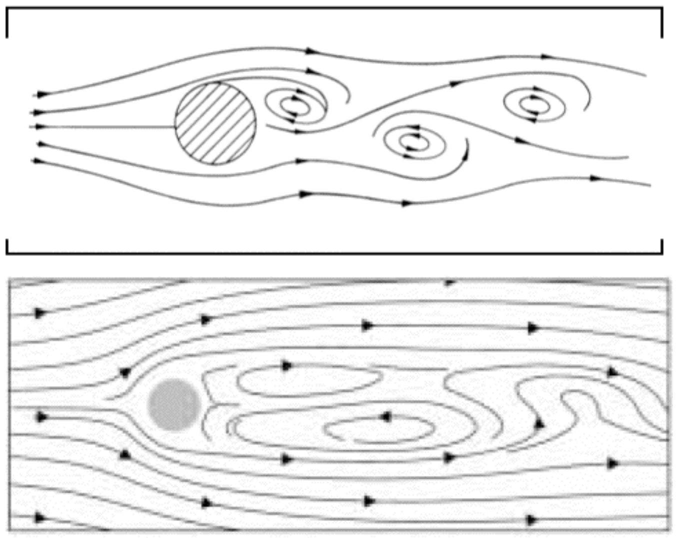


Figure 22: Top – hand-drawn illustration of flow around a cylinder [Feynman, 1964] that motivated the automatic illustrations of Turk and Banks shown on bottom.

This process uses an energy variance function to rate a blurred image of the streamlines. The length and placement of the streamlines are modified and rated in an iterative process until the image converges to an ideal representation. The iterative refinement approach is computationally expensive, because much time is spent performing changes that are later rejected by the energy variance test. Furthermore, because this process is image-based there is no obvious approach to augmenting the procedure for three-dimensional flow fields.

1.3.3 Evenly-Spaced Streamlines of Arbitrary Density

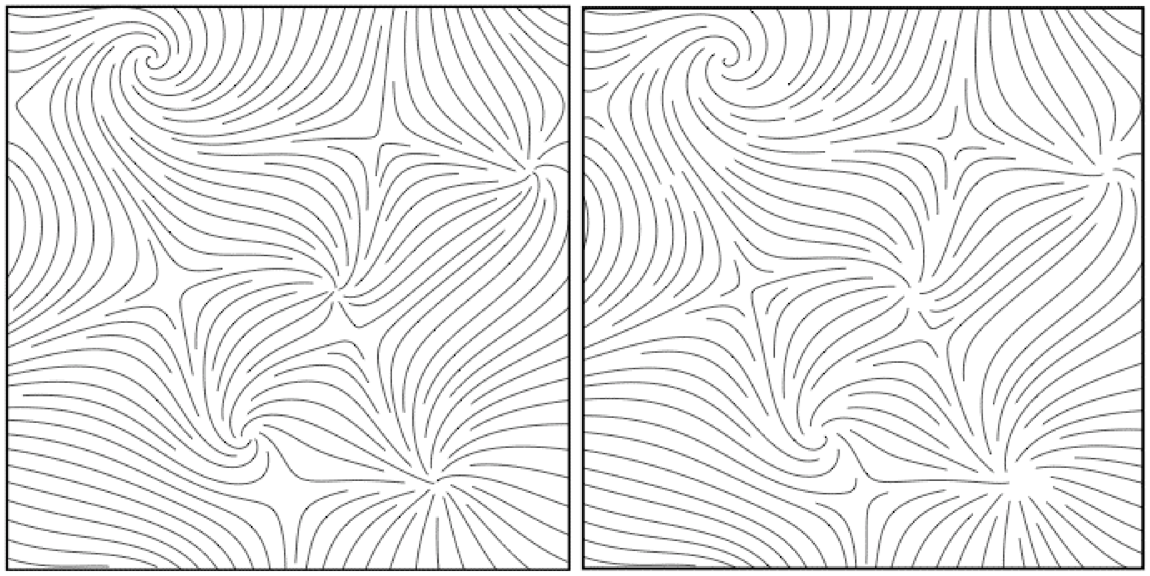


Figure 23: Streamlines created using the seeding strategies of Turk and Banks (left) and Jobard and Lefer (right).

Jobard and Lefer developed an alternative algorithm for producing the same results as Turk and Banks in a much more efficient manner [Jobard & Lefer, 1997]. This approach uses a user-defined minimal separating distance between streamline control points to define the density of the field. In order to control the distance between streamlines, a Cartesian grid that has a cell width equal to the separating distance is super-imposed over the domain, and streamlines are approximated by a set of equally spaced sample points along the line. Each grid cell contains a list of references to streamline control points located inside the cell. As each point along the streamline is calculated, the point is compared to the points referred to by the cell it passes through and the eight surrounding cells. If the distance separating the points is less than the minimal separation distance, then integration is stopped.

```
Compute an initial streamline and put it into the queue
Let this initial streamline be the current streamline
Finished := False
Repeat
    Repeat
        Select a candidate seedpoint at $d = d_{sep}$ apart from the current streamline
    Until the candidate is valid or there is no more available candidate
    If a valid candidate has been selected Then
      Compute a new streamline and put it into the queue
    Else
      If there is no more available streamline in the queue Then
        Finished := True
      Else
        Let the next streamline in the queue be the current streamline
      EndIf
    EndIf
Until Finished=True
```

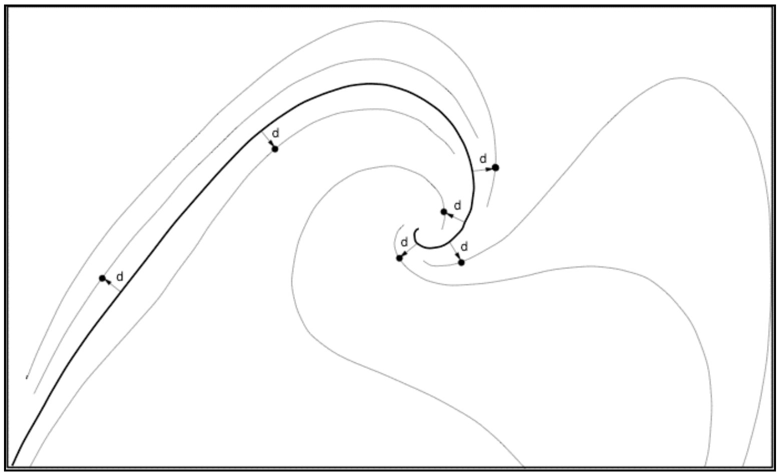


Figure 24: Top: pseudo code for the Jobard and Lefer streamline seeding algorithm. Bottom: the graphical results of one iteration; the dark line is the initial streamline.

Jobard and Lefer's process for creating evenly spaced streamlines is initiated by creating an initial streamline at a selected location and pushing the streamline on a queue. Then, for each control point of each streamline in the queue, a search is performed for a valid seed point. Seed points are created at the separation distance apart from control points orthogonal to the direction of the streamline at the control point. If a valid seed point is found, a streamline is created and inserted in the queue. This process repeats until no more valid seed points can be found. As a result, streamlines are created throughout the domain with a density that is determined by the separating distance.

In addition to performance over the streamline placement of Turk and Banks, Jobard and Lefer's method can easily be extended to three dimensions. In this case the grid becomes a series of cubes as opposed to squares and the distance test used in determining valid control points must test a maximum of twenty-seven cases as opposed to nine. Fuhrmann and Gröller confirmed the facility of such an implementation in their presentation of three-dimensional dash tubes [Fuhrmann & Gröller, 1998].

1.3.4 Stream Arrows

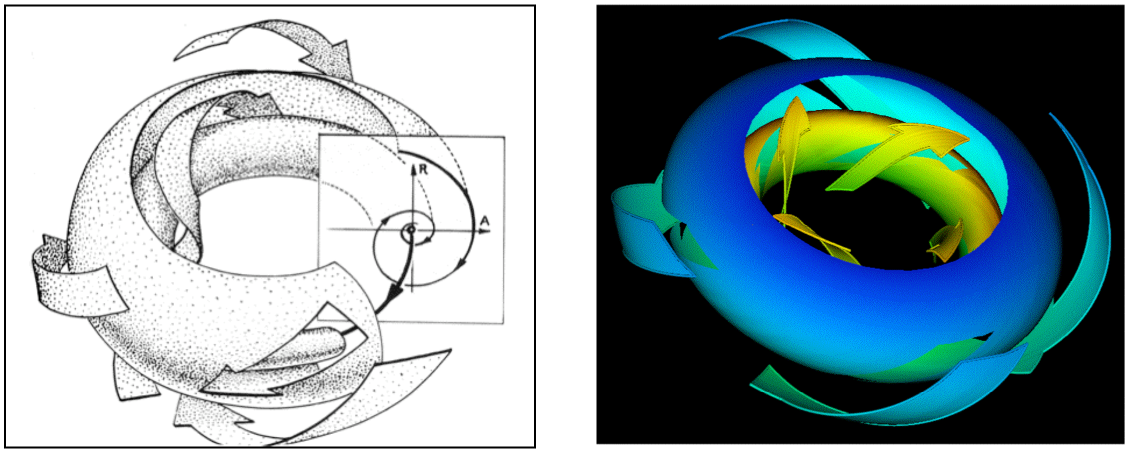


Figure 25: Illustration from a dynamics text [Abraham & Shaw, 1992] (left) that motivated the automatic illustrations of Löffelmann, Mroz and Gröller (right).

In [Löffelmann et al., 1997] Löffelmann, Mroz and Gröller introduced stream arrows, an extension to stream surfaces, motivated by illustrations found in a dynamic systems text (see Figure 25). Their method produces a series of tessellated transparent arrows that are texture mapped over a surface that is everywhere tangential to the flow field. These images extend traditional stream surface representations by providing directional cues and reducing occlusion as one may observe the behavior of overlapping surfaces through arrow-shaped windows. Stream arrows were constructed to be relatively short and evenly spaced, unlike the illustrations we sought to emulate.

1.4 Summary

In conclusion, all of the research listed above has successfully produced methods for visualizing flow fields motivated by images produced through hand-designed illustrations. However, none of the methods produce images that resemble ocean current illustrations we sought to emulate, shown in Figures 4, 5 and 6.

CHAPTER 2

OCEAN MODEL DATA

The data used as a basis for the research of this thesis was output from the finite-volume coastal ocean model (FVCOM) of the Gulf of Maine [Chen et al., 2002]. Two sets of data produced from this model were used. The data sets were modeled after the historical climatological properties in the Gulf of Maine in February and July.

The FVCOM model is computed using a finite element irregular triangular mesh on the horizontal x-y plane i.e., longitude and latitude, and using sigma-coordinates in depth along the z-axis. Sigma coordinates are valued between 0 at the surface and 1 at the ocean floor, thus representing position along the z-axis as a percentage of depth.

In order to make path tracing simpler and more efficient, we converted the irregular mesh of the model to a regular grid in universal transverse Mercator (UTM) coordinates. The UTM projection was calculated using the PROJ.4 Cartographic Projections library [Evenden, 2000].

In the Gulf of Maine, water particles trace out a spiraling path with loops on each tidal cycle. The data was averaged over 10 full tidal cycles (just over 5 days) of the model output in order to create a steady flow field that approximated the behavior of the field over that time period.

In the model data, longitude and latitude vector components (*u, v*) are represented in different units than the depth component (*w*): meters per second and sigma units per second, respectively. In order to facilitate path tracing the vector components were converted to a common unit of cell width per second, representing rate of travel through a voxel.

In two-dimensional representations, only the top layer of the model was used to illustrate surface currents. In three-dimensional representations the full volume of the model was used.

2.1 Horizontal Grid Conversion

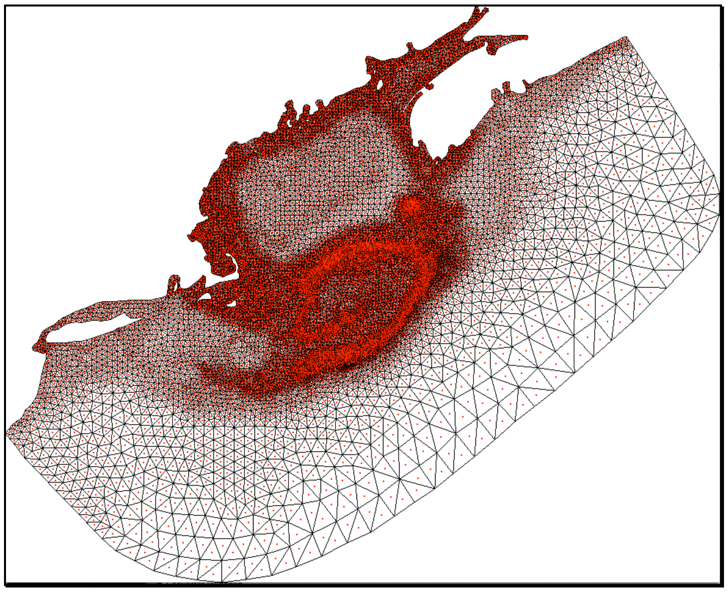


Figure 26: Triangular mesh (black) with centroids (red) indicating data point locations on the horizontal plane for the FVCOM model output of the Gulf of Maine.

The horizontal latitude and longitude domain of the model is subdivided into a set of non-uniform, non-overlapping triangular cells. Data values are stored at either the vertices of the triangle or at the geometric centroid (center of mass) of the triangle. There is one centroid per triangular cell and three vertices that are potentially shared with adjacent triangular cells. With this topology there are approximately twice as many vertices as there are centroids in the entire mesh. Figure 26 depicts the triangular mesh and centroids.

The following data values from the output of the model were used in the research for this thesis:

- *u, v, w* – flow vector components,
- H – mean water depth (bathymetry),
- ζ – free surface elevation (tide),
- θ – temperature,
- s – salinity.

The flow vector values are located at the triangle centroids and all other data values are located at the triangle vertices. Figure 27 illustrates the location of data values within a triangular cell.

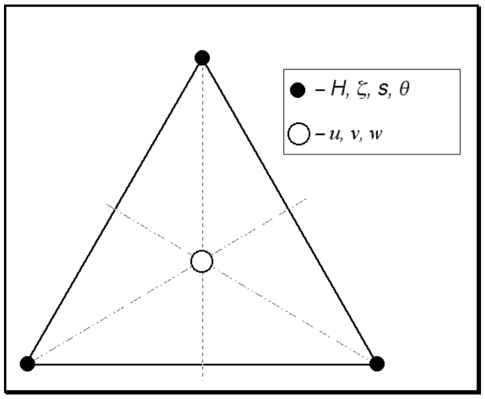


Figure 27: Arrangement of data values in a triangular cell. Adapted from [Chen et al., 2002]

The data values are interpolated such that the values are located at coordinates of a regular Cartesian grid in the horizontal plane, as shown in Figure 28. The data values that occur on the triangle vertices are interpolated by the following process:

1. select a Cartesian grid coordinate,

2. find the triangular cell in the mesh that encloses the coordinate, and
3. using the values at the triangle vertices and linear interpolation, calculate the value at the grid coordinate.

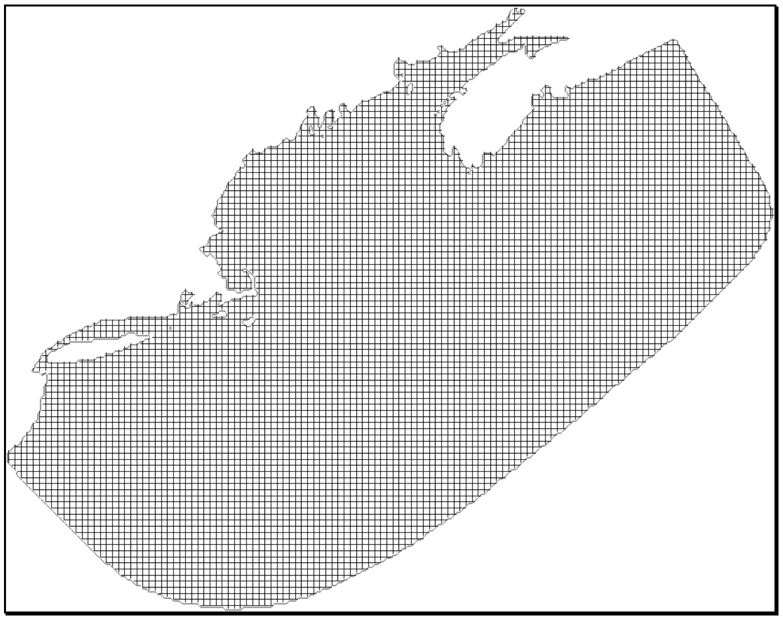


Figure 28: The regular grid produced after conversion.

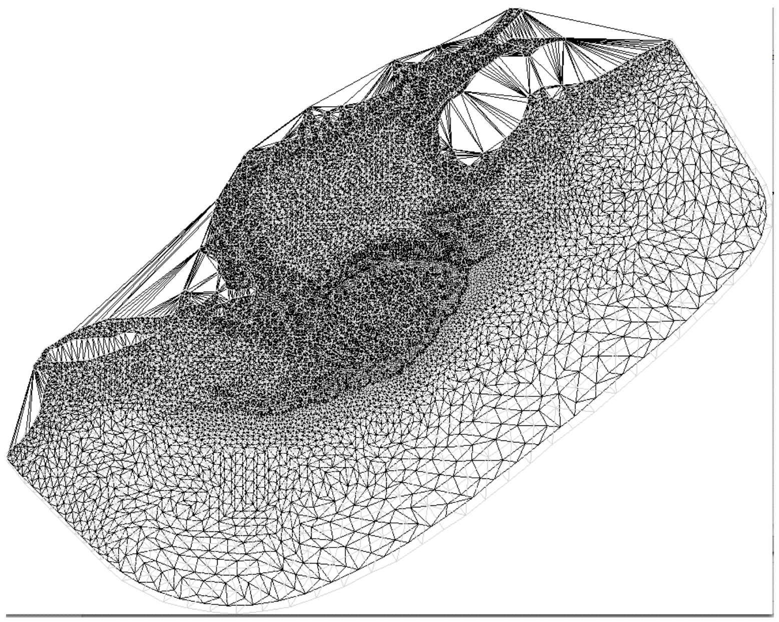


Figure 29: Delaunay triangulation of the centroids.

The centroids, however, are not arranged in a triangular mesh. However, a mesh can be generated from the centroid positions using a Delaunay triangulation [Shewchuk, 1996]. The values at the centroids are interpolated in a similar manner to the vertex values using the resulting mesh. Consequently, the Delaunay triangulation of the centroids creates triangular cells in areas where the original mesh does not. Note in Figure 29 where triangle cells from the centroid mesh intersect land areas, such as Cape Cod, Long Island and Nova Scotia. To avoid erroneous flow across land, vector components are only translated to the Cartesian coordinate if the point lies within the original mesh.

2.2 <u>Sigma Coordinates</u>

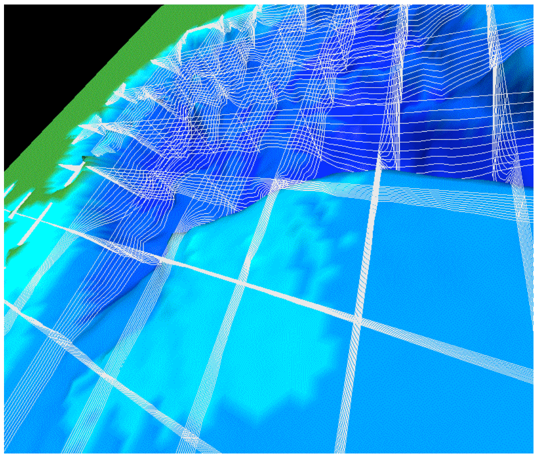


Figure 30: Sigma-coordinate grid (white) around George's Bank.

Sigma-coordinate models divide the depth range into a fixed number of layers, irrespective of the depth. Thus, each point on the horizontal grid has the same number of depth samples associated with it but these will be close to each other in shallow water and widely separated in deep water. The sigma-coordinate is valued between 1 and 0 and is defined by the following equation:

$$\sigma = (z + \zeta) / (H + \zeta)$$

where z is the physical distance of the coordinate from the mean surface. Therefore, σ is 1 at the bottom and 0 at the surface.

2.3 Data Averaging

The model data output for flow, temperature, salinity and surface elevation are arranged in a series of time steps one quarter of a tidal cycle apart (one complete tidal cycle is approximately twelve hours and twenty-five minutes). Figure 11, shown in Chapter 1, depicts a characteristic path of advection for this data output, which traces a looping path for every tidal cycle.

To approximate this behavior over time, the flow, temperature, salinity and surface elevation data were averaged for the number of complete tidal cycles. This process 1) allows for more manageable memory sizes at larger resolutions of data and 2) enables the use of streamlines to represent flow over a period of time.

In this averaging process it was found that the free surface elevation (ζ) values become negligible for our purposes and may be eliminated all together. Therefore, the sigma-coordinate equation is simplified to:

$$\sigma = z \,/\, H.$$

2.4 Homogenizing Vector Units

The vector components were converted to cell units per second simply by dividing the u and v components by the width of the Cartesian grid cell and the w components by the sigma value spacing between layers.

CHAPTER 3

ALGORITHM DESCRIPTION

The goal of this research is to automatically create images that expose major currents present in ocean model data in a manner that is similar to illustrative methods. The desired end result consists of variable-width ribbons overlaid with arrows that highlight areas of strong, coextending, parallel flows. This is accomplished through a streamline approach by collecting sets of parallel flowing streamlines in areas where there are strong continuous flows. The outermost points in this set of streamlines are used to calculate the vertices of a new object, which we refer to as a *super-streamline*. The algorithm for creating super-streamlines was developed such that it may be applied to two and three-dimensional data sets. Super-streamlines are drawn either as variable width ribbons or variable width tubes. Secondary currents are detected and shown as thin arrows.

3.1 Data Structures and Key Parameters

The primary data structures used in this algorithm are:

- *Streamline* – a list of equally spaced *control points* (defined below) that form a line tangential to the flow field.
- *Streamline queue* – used in calculating streamline placement.
- *Grid* – a Cartesian grid super-imposed over the domain to facilitate distance calculations between control points. The grid is created to be either two-

dimensional (in the *x-y* plane) or three-dimensional (*x, y, z*) following the dimensionality of the data. Each grid cell contains a list of streamline control points that are located within the cell.

- *Super-streamline* – a list of planar circles (or discs) defined by a center point, a normal (to the plane), and a radius. The planar circles are cross-sections of a variable-width tube.
- *Super-streamline list* – the final output of the algorithm. Graphical representations of super-streamlines depict currents.

Control points define the position and direction of streamlines in either two or three-dimensional space. In two-dimensions, positions have only *x* and *y* coordinates, representing longitude and latitude respectively, and vectors have only *u* and *v* components. In three-dimensions, depth position is represented by a *z* coordinate and vertical displacement is represented by a *w* component. Control points also contain values that identify the streamline and super-streamline the control point is a member of.

The fields of a control point are described below:

- *Physical coordinates* – used to calculate real distance (in meters) along a streamline,
- *Computational coordinates* – used to control spacing (in grid cell units) between control points,
- *Interpolated vector* – local direction (in grid cell units) of flow along the streamline at the control point position,

- *Streamline ID* – for access to the streamline the control point is on and its properties, such as: length, average speed, etc.,
- *Super-streamline ID* – used during the construction of super-streamlines to determine membership of a control point to a specific set of streamlines.

The following parameters govern the construction of the data structures listed above:

- *Separation distance* – defines the dimensions of grid cells and the distance from existing control points that new streamlines are seeded.
- *Sample length* – the distance between adjacent control points on the same streamline.
- *Minimum separation distance* – the minimum distance between streamline control points at which streamline integration is stopped.
- *Maximum separation distance* – the maximum radius of a super-streamline cross-section.
- *Minimum streamline length* – the minimum length of a valid streamline.

3.2 <u>The Basic Process</u>

Initially, a *principal streamline* is determined by calculating the longest and strongest streamline possible in the flow field using average speed and total distance along a streamline as a heuristic. A set of *neighboring streamlines* is generated around the principal streamline using a seeding method extended from Jobard and Lefer's evenly spaced streamline seeding method [Jobard & Lefer, 1997]. A *super-streamline* is constructed by calculating circular planes defined by the direction along the principal streamline and the surrounding neighbor streamlines. The circular planes define cross-sections of variable width tubes, which may also be drawn as variable width ribbons. To find additional super-streamlines, the entire process repeats with the stipulation that no point along a newly calculated streamline may occur in an area covered by a previously calculated super-streamline. The algorithm is summarized in pseudo-code below.

```
// Creates a set of super-streamlines
Repeat
    For each cell in grid
    {
     generate streamline such that no point on
       streamline is covered by a super-
       streamline
    }
    select longest-strongest streamline generated
      as principal streamline

    neighbour streamlines := calculate list of
      streamlines proximal and parallel to
      principal streamline

    new super-streamline := generate super-
      streamline from principal streamline and
      neighbor streamlines

    Add new super-streamline to super-streamline
      list

Until (no new principal streamline can be found)
```

Figure 31: Pseudo-code for creating a set of super-streamlines.

3.3 Generating the Principal Streamline

The objective of calculating the principal streamline is to find the streamline in the domain that best represents the overall shape of the strongest current. Ideally, the principal streamline is found to run near the center of the current for the entire length of the current. After experimentation with many heuristics, it was found that the streamlines that had the greatest calculated value of average speed multiplied by length best exhibited these features and, therefore, is used as a basis for calculating a principal streamline.

Because of this property, we also refer to the principal streamline as the *longest-strongest* streamline.

Figure 32 shows the experimental results that motivated the use of average speed multiplied by distance as a means for calculating principal streamlines. In this figure, the color-coding visually exposes channels of flow. The greatest valued streamlines in a channel appear near the channel center and follow the general contour of the channel.

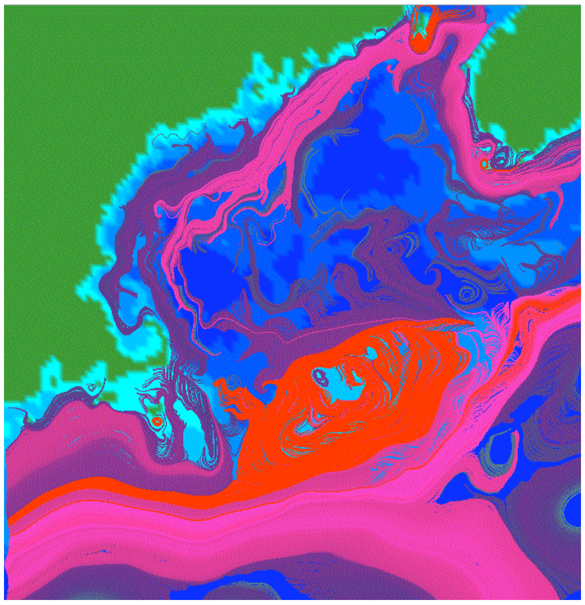


Figure 32: Densely seeded streamlines color-coded in descending order of "longest-strongest" from red to hot pink, to purple, to grey.

To find the so-called longest-strongest streamline in the domain, a streamline is seeded at the center of every grid cell and integrated forward and backward until the streamline leaves the flow field or until it is less than the minimum separation distance from a control point in the grid. After the principal streamline has been determined, it is inserted to the empty streamline queue and its control points are entered into the grid. The figure below shows the results of computing the initial principal streamline.

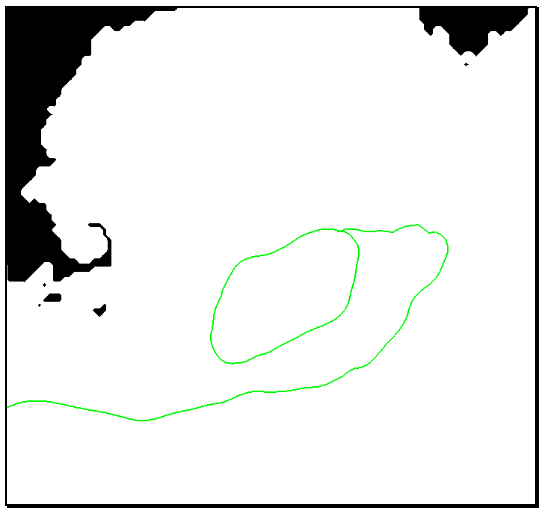


Figure 33: The initial principal streamline (green).

3.4 Generating Neighboring Streamlines

The principal streamline serves as an initial streamline in a streamline seeding process similar to the method described in [Jobard & Lefer, 1997]. This seeding method was chosen because it can handle both two and three-dimensional representations.

3.4.1 Streamline Seeding

In this process, each newly computed streamline is appended to a queue starting with the principal streamline. For each streamline in the queue, *candidate seed points* are generated at the separation distance apart from each control point. The principal streamline is used to initiate the seeding process. All streamlines created from the principal streamline (or successor streamlines) are appended to the queue. When all possible candidate seed points have been generated from the principal streamline, the next streamline in the queue becomes the "current" streamline. This process repeats until all of the streamlines in the queue have been explored and no more streamlines can be generated.

Candidate seed points are positioned such that they exist on the plane that is orthogonal to the direction at the control point on the current streamline. Candidate seed points are also positioned such that they are equally spaced in degrees of arc about the control point on the current streamline. In two-dimensional representations, two candidate seed points are created 180º apart; in three-dimensions, they are created 45º apart. Figure 34 illustrates the candidate seed points generated around a control point of the current streamline.

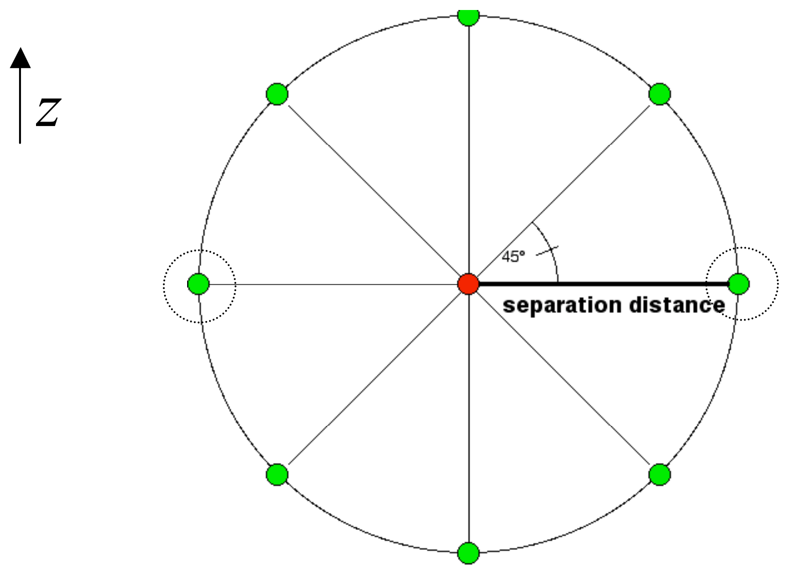


Figure 34: Candidate seed points (green) equally spaced around a control point of the current streamline (red). The circled seed points are those generated for two-dimensions. The direction of the current streamline points directly out of the page and the direction of the positive z-axis is shown on the left.

3.4.2 <u>Streamline Integration</u>

From each candidate seed point a neighboring streamline is generated. Neighboring streamlines are created by integrating forward and backward until one of the following conditions is met:

- the streamline has left the domain,

- a control point q on the neighboring streamline is less than the minimum separating distance from any other control point in the grid; i.e., any other control point belonging to any other streamline regardless of super-streamline membership,
- a control point q on the neighboring streamline is greater than the maximum separating distance, *dmax*, from the closest point p on the principal streamline,
- $cos^2(\theta) < |q - p| / dmax$;

where θ is the angle between the flow vectors at p and q. This has the effect of accepting more diverging or converging streamlines close to the principal streamline and enforcing more parallel flowing streamlines farther from the principal streamline. That is, as points on a neighboring streamline get further away from the principal streamline, they are subject to a stricter interpretation of "parallel". This equation was determined through trial and error experimentation.

Furthermore, a neighboring streamline may be rejected after calculation if its overall length is less than the defined minimum. If a valid neighboring streamline is produced, its control points are inserted into the grid.

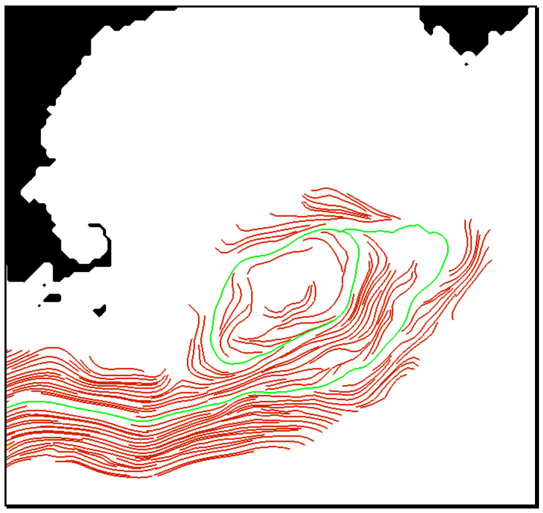


Figure 35: Neighboring streamlines (red) generated around the principal streamline (green).

3.5 Generating Super-Streamlines

The process of generating a super-streamline creates a list of planar circles (each defined by a center, a radius and a normal vector) from a principal streamline and its associated neighboring streamlines. The planar circles define cross-sections of a variable-width tube in three-dimensions or a ribbon in two-dimensions that encloses the principal and neighboring streamlines and follows the generalized contour of the streamlines' paths. In essence, generating a super-streamline captures the shape and contour of a current into geometrical object.

3.5.1 Defining the Plane

Circular planes are calculated for every control point along the principal streamline. The direction and location of the *current control point* define the position and orientation of the plane. The equation for the plane may be expressed implicitly as:

$$(N \bullet p) + D = 0,$$

where *N* is the direction of the streamline at the control point, *p* is its position, and *D* is the calculated plane constant. For two-dimensional representations the z-component of *N* will always be valued at zero and the z-component of *p* is constant.

3.5.2 Calculating Center and Radius

The remaining problem is to calculate the co-planar center point and radius of a circle that encloses the surrounding neighboring streamlines. This is accomplished by finding the points of intersection between the neighboring streamlines and the planar circle, such that the initial radius of the circle is the maximum separation distance and the initial center is the current control point. In two-dimensions, the points of intersection exist on a common line defined by the intersection of the x-y plane and the plane defined by the current control point.

The new center and radius are determined by calculating the minimum area circle that encloses all of the intersection points. The method presented by Gärtner for extremely fast and robust computing of the smallest enclosing sphere given a list of points was used for this application [Gärtner, 1999]. If none of the neighboring streamlines intersect the plane, the current control point defines the center and the minimum separation distance is used as the radius.

3.5.2.1 Reducing Overlap

In areas where there are spirals and sharp bends in the principal streamline two or more cross sections may capture the same area. This effect causes overlapping sections of the resulting ribbons or tubes. For example, a point further down the principal streamline from the current point may intersect the plane at a distance less than the maximum separating distance. The resulting tube would overlap itself in this area.

To overcome this problem, a simple distance test is performed on each of the neighboring streamline – planar circle intersection points during the calculation of the cross-section. If the principal streamline intersects the plane (other than the point that defines the plane) then those intersection point(s) are stored in a separate list. As each intersection produced from neighboring streamlines the distance to each principal streamline intersection is calculated. If any of these distances is less than the distance from the neighboring streamline intersection to the center of the planar circle, the intersection is rejected and, hence, not considered in the minimum enclosing circle calculation.

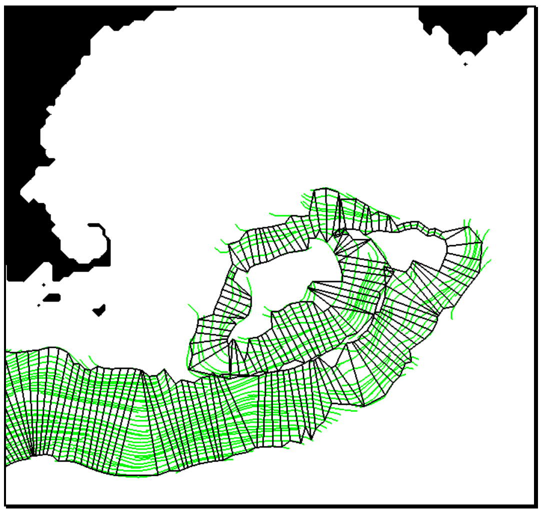


Figure 36: Wire-frame of a super-streamline (black) constructed from underlying streamlines (green).

3.5.2.2 <u>Reducing Folds</u>

Large, instantaneous bends in the principal streamline may also create unsightly folds in the super-streamline when successive planes intersect. To avoid this problem, a cross-section is rejected if it intersects the last cross-section inserted in the super-streamline.

3.5.2.3 Ensuring Convex Polygons

Super-streamlines may be rendered as a variable-width tubes or ribbons. The ribbons are defined as a strip of quadrilaterals, i.e., a list of left-right point pairs. A quadrilateral on the strip containing an angle greater or equal to 180º i.e., the quadrilateral is concave, will produce undesired effects when shading the ribbon. To compensate for this behavior, an additional cross-section is added to the super-streamline such that the ill-formed quadrilateral is bisected at the greatest angle, thus creating two triangles. Thus every polygon that occurs on the super-streamline is convex. Inserting a new cross-section into the super-streamline creates the bisection. Although the new cross-section does intersect the previous and following cross-sections at their edges, the bisection is performed after the test to reduce folds, thus these intersections are considered valid. Figure 37 demonstrates how a quadrilateral is bisected.

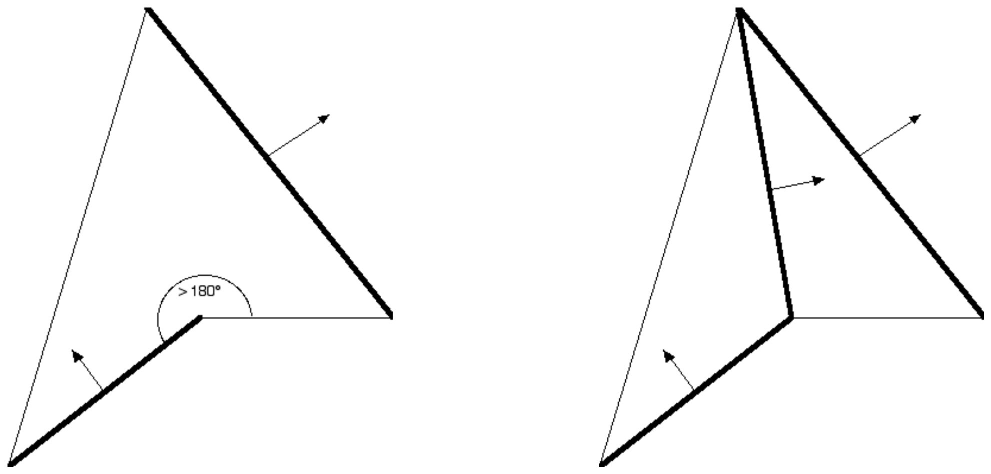


Figure 37: On left – an undesirable convex quadrilateral created by two consecutive cross-sections (shown as dark lines). On right – the result of adding a cross-section to bisect at the greatest angle.

3.5.3 Summary

The algorithm for calculating a super-streamline is summarized in the pseudo-code shown in Figure 38.

```
// Creates a super-streamline given a principal streamline and a list
// of neighboring streamlines

For each control point p in the principal streamline
{
    center := p.position
    radius := Maximum separation distance
    plane := createPlane(p.direction, center)

    // Compensate for possible overlap
    For each point i along the principal streamline that intersects
      with plane
    {
      If ( |center – i| < radius*2 and i is not center) Then
        Add i to principal intersection list
      End If
    }

    // Calculate enclosing circle
    For each point j on the neighboring streamlines that intersects
      with plane
    {
      If ( |center – j| < distance between j and all points in
           principal intersection list ) Then
        Add j to neighboring intersection list
      Endif
    }

    If (size of neighboring intersection list = 0 ) Then
      radius := minimum separation distance
    Else
      circle := calculate smallest enclosing circle from neighboring
        intersection list
      radius := circle.radius
      center := circle.center
    Endif

    clear intersection lists

    If (circle does not intersect previous circle) Then
      If (circle and previous circle create a concave quad) Then
        Calculate new circle to bisect the quad and add new circle to
            super-streamline
      Endif
      Add circle to super-streamline
      previous circle := circle
    Endif
}
```

Figure 38: Pseudo-code for computing a super-streamline.

3.6 Smoothing

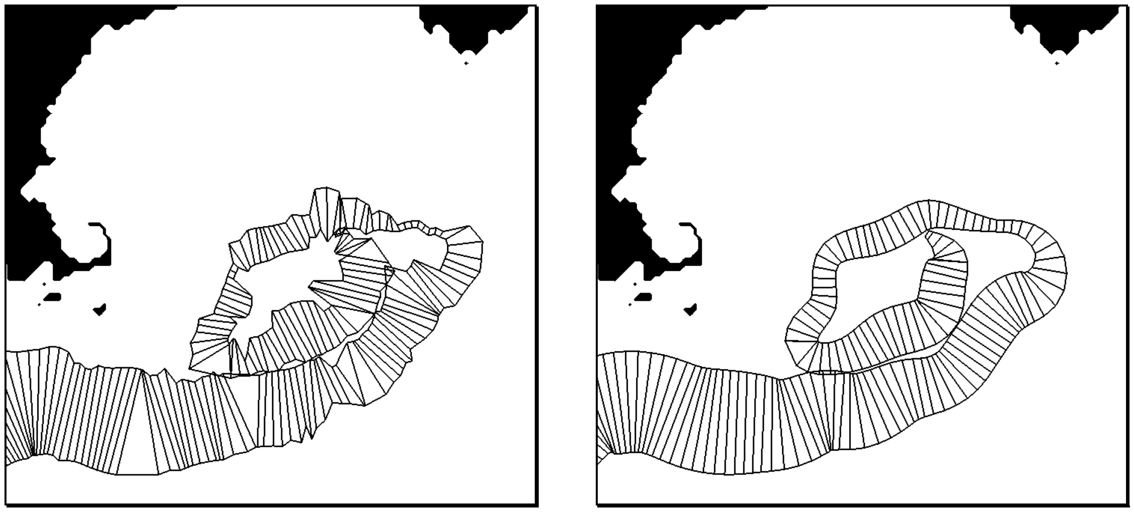


Figure 39: Left – wire-frame of a super-streamline before smoothing; Right – after smoothing

Once all of the vertices of a super-streamline have been computed, the structure may be smoothed to create a more aesthetic and illustrative shape as demonstrated in Figure 39. This process has four primary components:

1) blend radius values,
2) recalculate the list of center points as a continuous curve,
3) recalculate the plane normals based on new center points, and
4) eliminate folds and concave polygons in newly generated cross-sections, as described in the previous section.

The smoothing process may be repeated until the desired effect is obtained.

3.6.1 Blending Radius Values

Unsmoothed super-streamlines often exhibit areas where radius values vary greatly between successive values. When rendered in this form, either as ribbons or as tubes, the edges of the super-streamline appear jagged. To create smoother transitions between successive values, each radius value is averaged with the values before and after it in the super-streamline list. Thus, each new radius value is calculated using the following equation:

$$r_i' = (r_{i-1} + r_i + r_{i+1}) / 3$$

where r_i is the original radius value and r_{i-1} and r_{i+1} are the values that occur before and after the original value, respectively.

3.6.2 Recalculating Center Points

Lines defined by successive center points of unsmoothed super-streamlines are very irregular, i.e., they follow a jagged path. Recalculating the positions of the center points using cubic spline interpolation greatly improves the overall smoothness of the resulting super-streamline. In this process, a subset of the center points is defined by sub-sampling the center points at a regular interval, e.g., every third point along the super-streamline. Cubic spline interpolation is then used to generate a number of points that is equal or greater than the number of points along the original unsmoothed super-streamline. Blended radius values may then be interpolated for each new center point using the same process. The algorithm used for cubic spline interpolation was adapted from [Press et al., 1992].

3.6.3 Recalculating Plane Normals

The plane normals at the newly computed center points may be easily calculated from the positions of the previous and next center points on the new, smoothed super-streamline. More precisely, the new plane normal at a specific center point is the normalized vector created from the previous center point to the next center point. This may be expressed in the equation:

$$N_i = (C_{i+1} - C_{i-1}) / | C_{i+1} - C_{i-1}|$$

where C_{i-1} and C_{i+1} are the previous and next center points produced from the cubic spline interpolation.

3.6.4 Eliminating Folds and Concave Polygons

The three previous processes create a new list of planar circles (or cross-sectional disks). The resulting disks, however, may be subject to the same pitfalls described in the previous section regarding the creation of super-streamlines. Thus, as in the creation process, successive disks that intersect are rejected to eliminate folds and concave quadrilaterals are bisected to ensure proper shading of ribbons.

3.7 Final Rendering

Super-streamlines are structured such that they may be easily rendered either as variable-width ribbons or tubes. The center point, plane normal, and radius are used to calculate the vertices that define the outer boundaries of the ribbons and tubes. In both representations all points generated are located on the plane and are equidistant from the center point as defined by the radius value.

At the rendering stage of both ribbons and tubes, an additional vertex is added to each end of a super-streamline. The new vertices occur one radius distance from the initial end points and are positioned before and after the super-streamline according to the direction of the flow. The addition of these vertices serves two purposes: 1) to increase the probability of super-streamlines intersecting where currents branch and merge, and 2) to give a less abrupt appearance where currents start and end.

3.7.1 Ribbons

When rendered as ribbons, two points are generated for each circular plane such that the line between the two points is orthogonal to both the plane normal and the positive z-axis. In two-dimensions, this simply means that all points exist on the same x-y plane. The resulting list of point pairs is rendered as a strip of quadrilaterals to produce a variable-width ribbon. Ribbons are enhanced with illustrative details in a straightforward manner by adding color-coding, edge lines, and arrow glyphs.

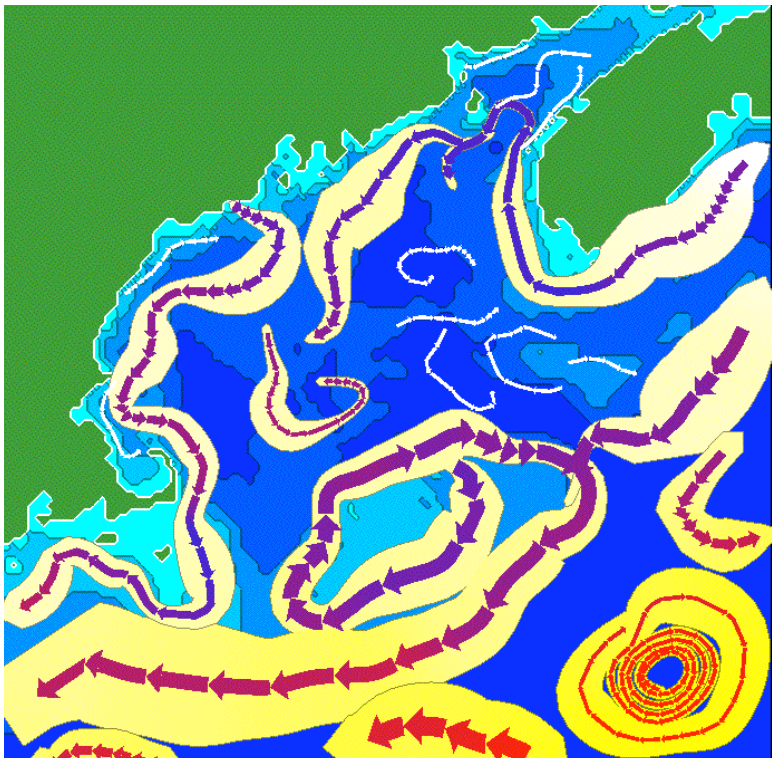


Figure 40: Automatic illustrations of 2D surface flow in the Gulf of Maine - ribbons overlaid with arrows, minor currents shown by thin white arrows.

Figure 40 shows a two-dimensional super-streamline illustration using ribbons. The ribbons are color-coded between yellow and white to represent salinity levels. The ribbons are overlaid with wide arrows color-coded between red and magenta to indicate temperature values. The width of the arrows is a function of the average width of the underlying ribbon. The length of the arrow correlates to the velocity of the flow. Secondary currents are determined by a user-defined minimum average width and are represented solely as thin arrows.

Figure 41 shows the techniques described above applied to super-streamlines produced from three-dimensional data. Because of the limitations of representing a three-dimensional image on a static two-dimensional medium and the tendency for super-streamlines to occlude each other, only the first three super-streamlines generated are shown.

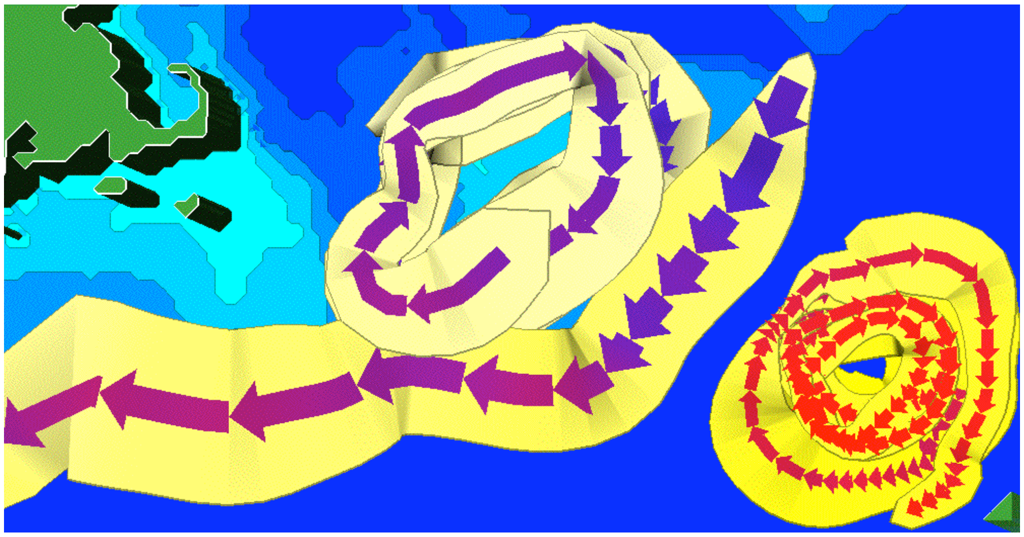


Figure 41: Automatic illustrations of 3D currents in the Gulf of Maine limited to the first three "longest-strongest" currents.

3.7.2 <u>Tubes</u>

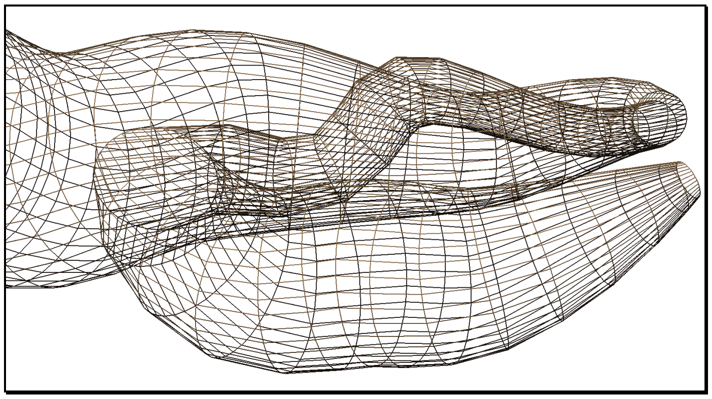


Figure 42: Wire-frame of a super-streamline tube.

When super-streamlines rendered as tubes, many points are generated for each circular plane such that the points are equally spaced radially about the center point. A greater number of points generated increases the resolution of the rendered tube. The points generated by two successive circular planes define a segment of a tube. A tube segment is rendered as a strip of quadrilaterals by traversing the perimeter of both circles in unison. Figure 42 shows the resulting structure in wire-frame.

Although the images produced by rendering super-streamlines as tubes do not resemble the illustrations we sought to emulate, tubes are an effective method representing three-dimensional current flow. Thinning the width of the tube reduces the occlusion issues inherent in these images. Thinning is accomplished by calculating the vertices of the tube as a percentage of the radius from the center. Figure 43 demonstrates the results of thinning tubes.

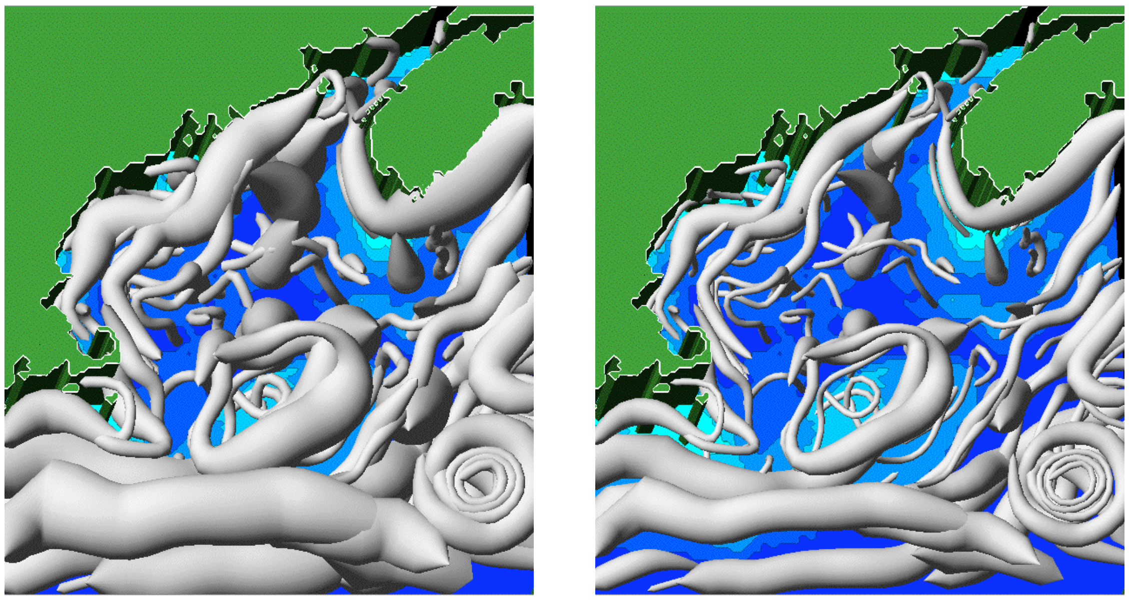


Figure 43: Three-dimensional super-streamlines drawn as shaded variable width tubes: Left – full width, Right – 60% thinning

3.8 Summary

This chapter has introduced a method for automatically generating images of ocean currents either based on three-dimensional volumetric data or two-dimensional surface data. The process for creating the images consists of creating a list of geometrical objects called super-streamlines, smoothing their shape, and rendering them as a series of variable-width ribbons or tubes.

The algorithm for creating super-streamlines was developed such that it may be applied to both two and three-dimensional data sets. This was accomplished by treating two-dimensional vector fields as a special case of three-dimensional vector fields, i.e., the two-dimensional vector field exists on a horizontal plane within a three-dimensional space.

For illustrative effect, ribbons are rendered with dark edge lines, color-coded to temperature and salinity values, and overlaid with arrow glyphs. To represent minor currents, ribbons that have an average width less than a user-defined minimum are drawn solely as thin arrows. Super-streamlines may alternatively be drawn as a series of variable-width tubes when created from three-dimensional flows. To reduce occlusion, tubes may be thinned as a percentage of their width.

Although efficiency was not a goal of this research, the time to execute the algorithm on a 667 MHz G4 processor with 768 MB RAM was measured. The time execute the algorithm on a 2D data set with a 201 row by 255 column grid was approximately 2.5 minutes. The time to execute the algorithm on a 3D data set with a 201 row by 255 column by 30 layer grid was approximately 12.7 minutes. As is discussed in the concluding chapter, improving the efficiency of the algorithm is a future research objective.

A gallery of sample results is shown in the following section. These illustrate current patterns in the Gulf of Maine that are modeled from recorded climatological conditions in February and July. Illustrations of both surface currents and full volume currents are shown at various zoom levels and view points.

3.9 Sample Results

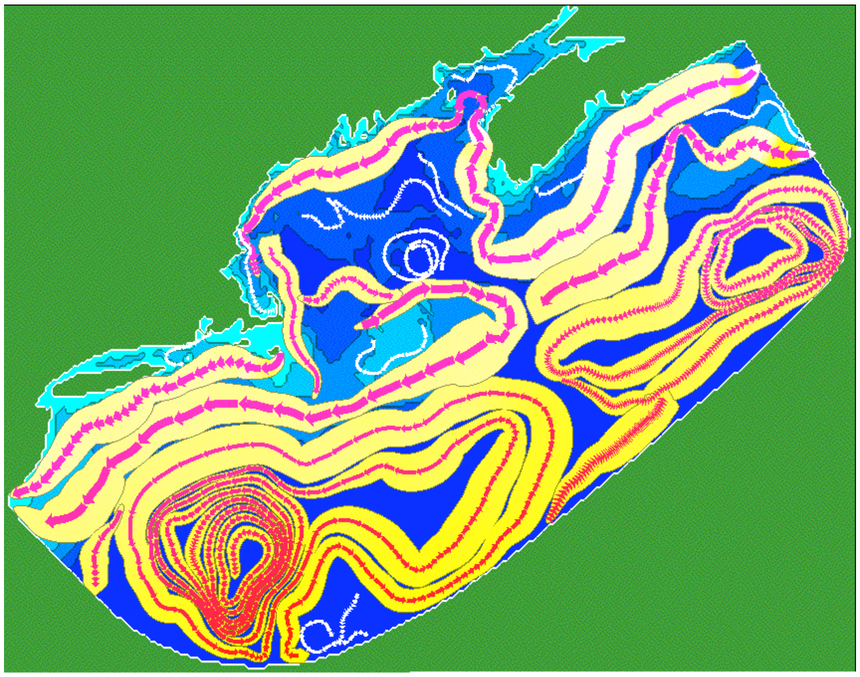


Figure 44: Automatically generated illustration of Gulf of Maine “February” data set surface layer

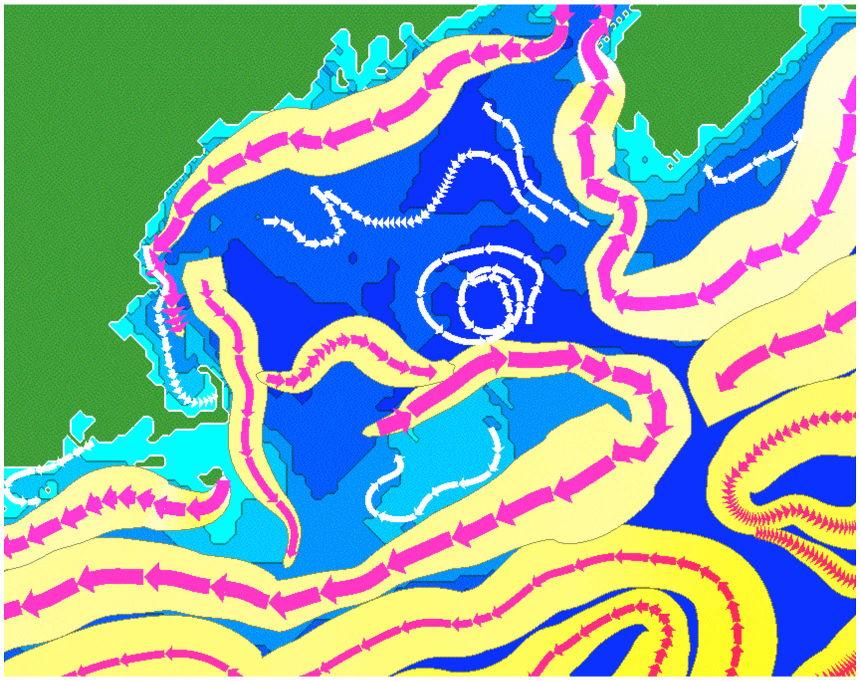


Figure 45: “February” surface layer – zoom on center

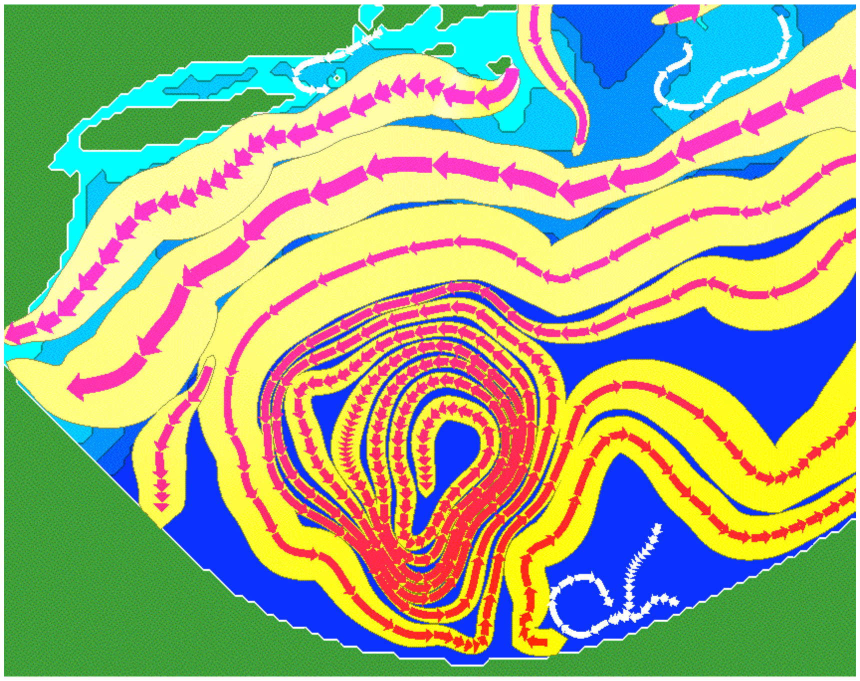


Figure 46: “February” surface layer – zoom on southwest corner

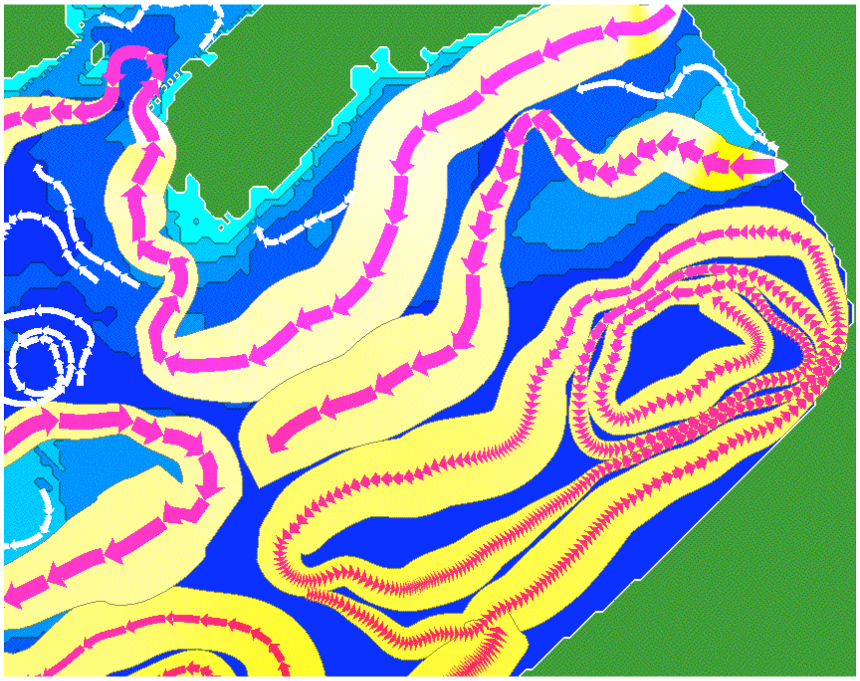


Figure 47: “February” surface layer – zoom on eastern corner

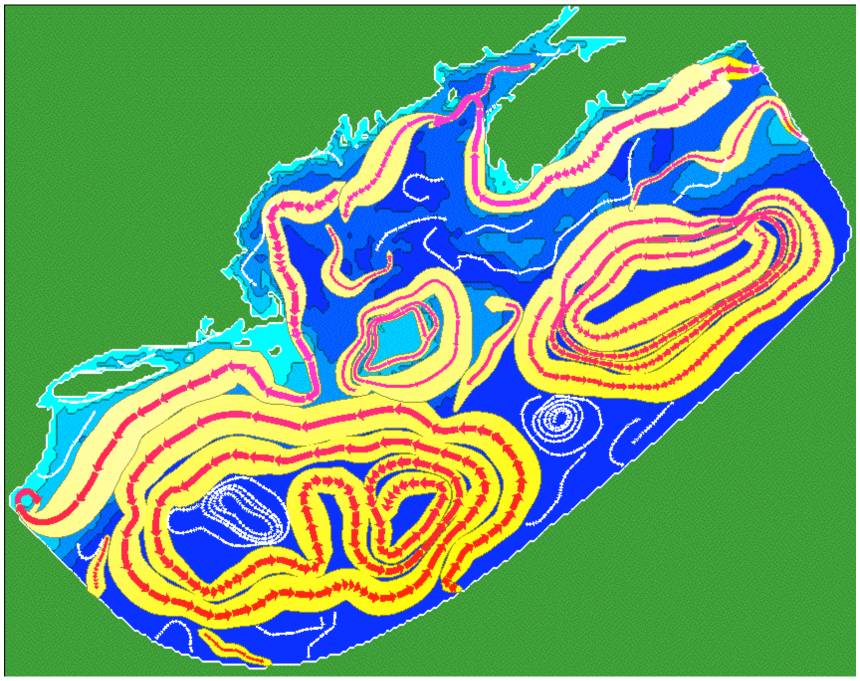


Figure 48: Automatically generated illustration of Gulf of Maine "July" data set surface layer

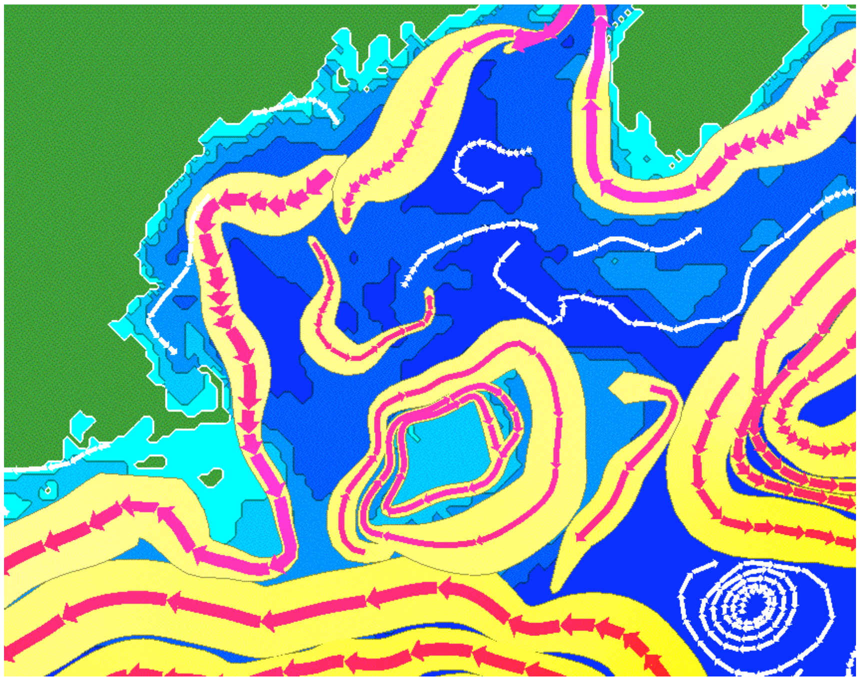


Figure 49: "July" surface layer – zoom on center

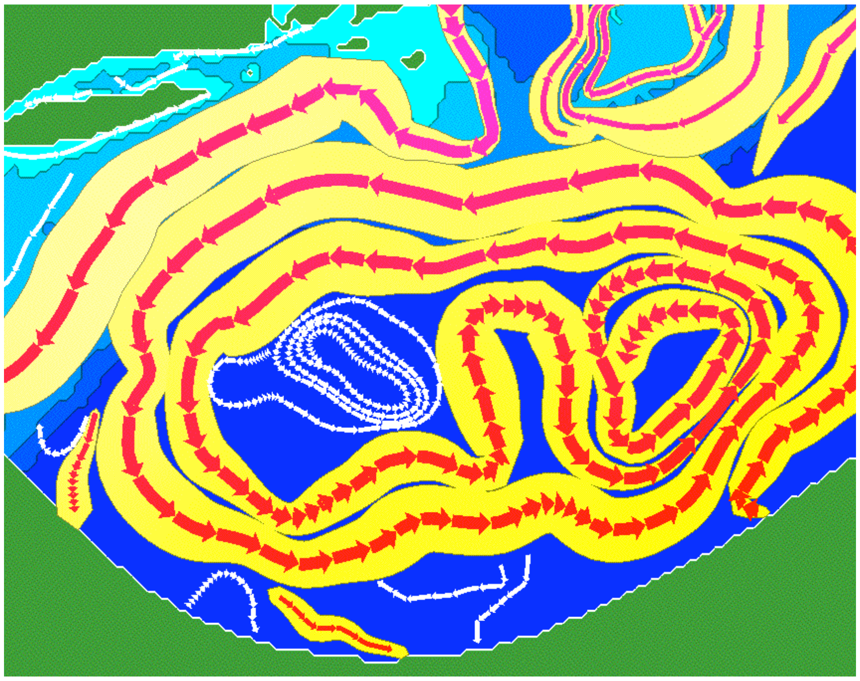


Figure 50: “July” surface layer – zoom on southwest corner

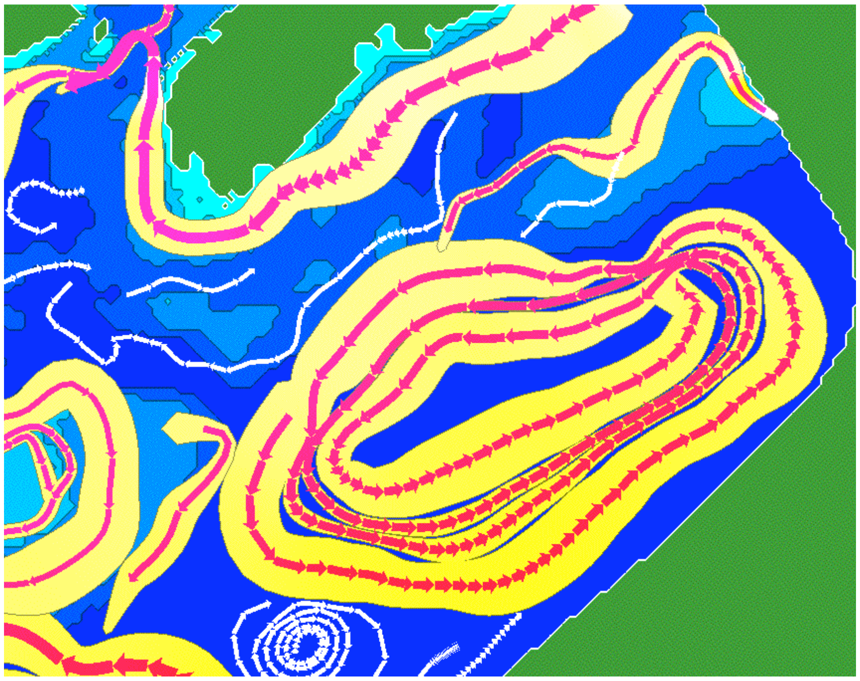


Figure 51: “July” surface layer – zoom on eastern corner

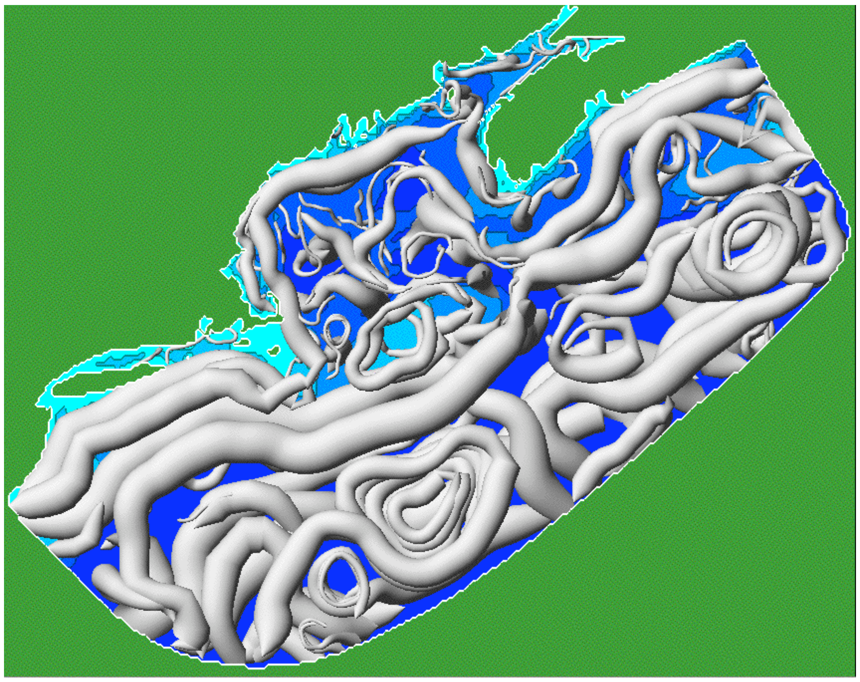


Figure 52: Tubes 60% thin – "February" data set

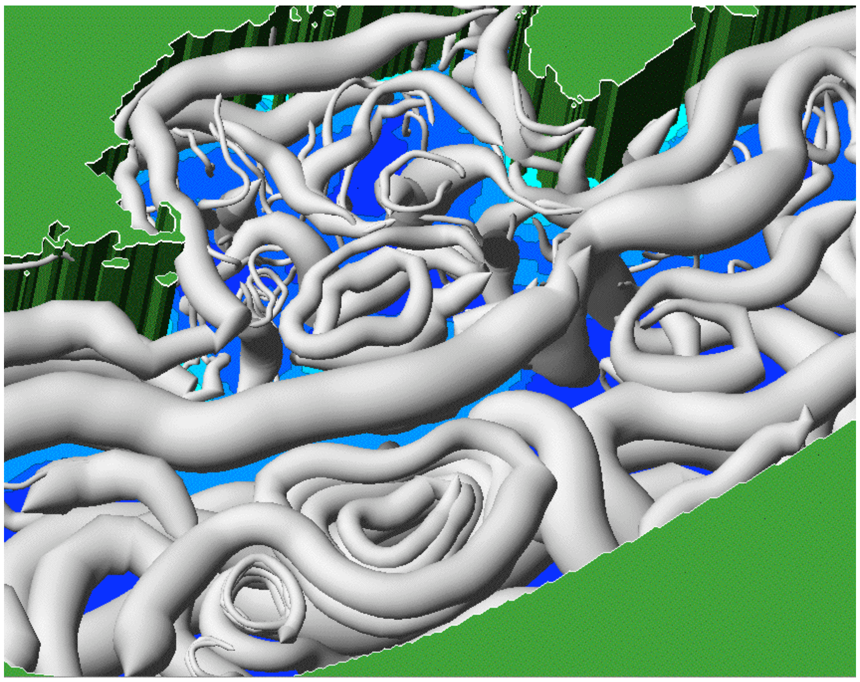


Figure 53: Tubes 60% thin – "February" data set. Zoom on center and rotate.

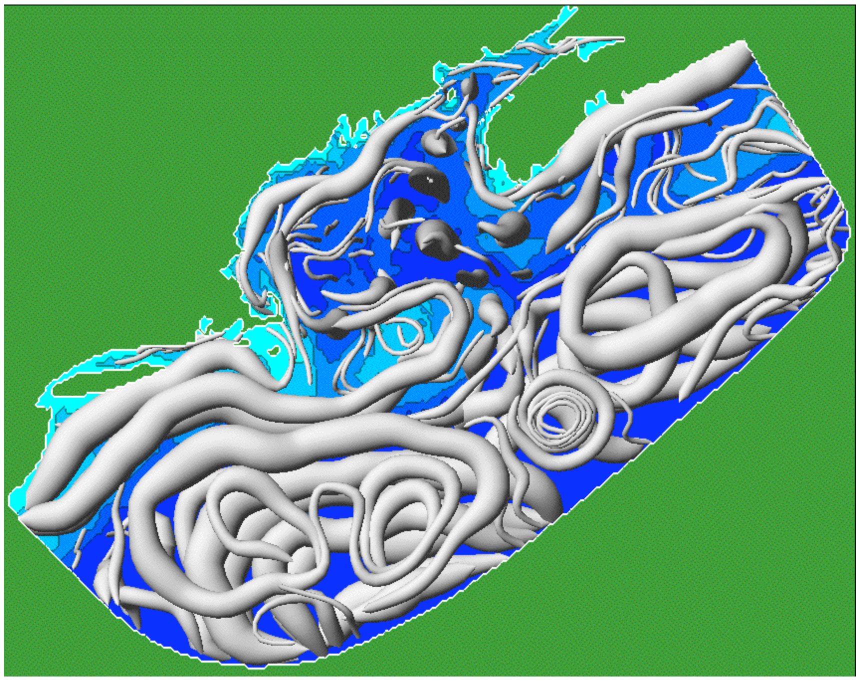


Figure 54: Tubes 60% thin – “July” data set

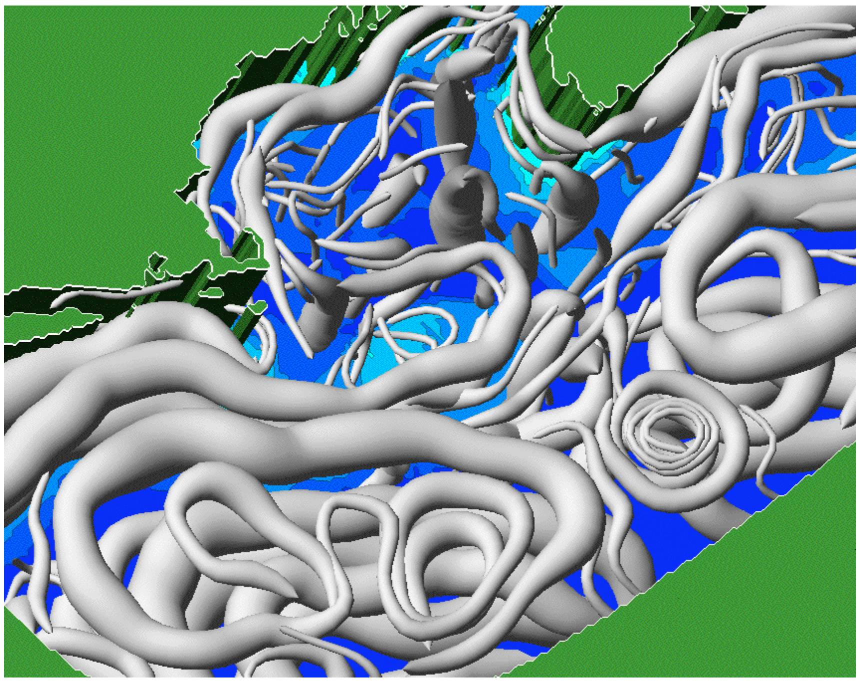


Figure 55: Tubes 60% thin – “July” data set. Zoom on center and rotate.

CHAPTER 4

CONCLUSION

The goal of this research was to design an algorithm for automatically generating images of ocean currents that 1) are created from ocean model data, and 2) have the aesthetic and demonstrative qualities inherent to illustrations. Ultimately, the images created by the algorithm provide an abstract overview depicting the behavior of a flow field over time by emphasizing areas where strong, coherent currents occur. Illustrations found on oceanographic websites, such as those shown in Figures 1, 4, 5 and 6, served as a basis for the design of the images produced from the algorithm. In particular, we sought to emulate the following features:

- smooth, broad strokes that indicate the paths of major currents
- width of the strokes are proportional to the area occupied by the current
- straightforward indications of direction and strength of flow
- color-coding according to ancillary attributes
- clear indications of branching and merging of currents
- thin arrows representing minor currents.

4.1 <u>Research Overview</u>

The solution presented in this thesis is to create a series of geometrical objects, called *super-streamlines*, which capture the shapes and contours of currents based on areas where strong, coextending, parallel flows exist. A super-streamline is created by first finding the longest and strongest streamline in the domain, called the *principal streamline*, and then seeding in the local area around the principal streamlines to find a set of streamlines, called *neighboring streamlines*, that run along to the principal streamline. The vertices of a super-streamline are then defined by the outermost points along the set of neighboring streamlines that exist orthogonal to the direction of flow along the principal streamline. A grid structure is employed to control intersections and spacing between multiple super-streamlines.

The algorithm was applied to the output of a finite volume model of flow patterns in the Gulf of Maine simulating climatological conditions in February and July. The horizontal flow components along the surface layer of the model were used to create two-dimensional illustrations. Three-dimensional illustrations were created from the entire volume represented by the model.

To create the desired illustrative features, a specialized rendering process was applied to the super-streamlines created from the ocean model data. Rendering the shape of the super-streamlines as a strip of consecutive quadrilaterals, or ribbons, emulates smooth, broad strokes. The width of the ribbon varies along its length as to indicate the area occupied by the current. Minor currents are defined by the mean width of a super-streamline. If the mean width is less than a user-defined minimum width value, the super-streamline is rendered solely as a series of thin arrows.

To indicate the direction and strength of the flow, a series of arrow glyphs are overlaid on the ribbons. The width of the arrows is proportional to the mean width of the ribbon and the length of the arrows is proportional to the strength of the current. Color-coding of salinity and temperature values are applied to the ribbons and arrows, respectively.

Ribbons are extended at both ends beyond the computed length of the super-streamlines in order to increase the probability of intersection in areas of branching and merging. Although this approach does not directly represent branching and merging of currents, the phenomena are implied by the intersection and direction of the ribbons.

The illustrative rendering process described above may be applied to super-streamlines constructed from both two and three-dimensional flows. However, the ribbons constructed from three-dimensional flow suffer from perceptual issues, such as occlusion and poor indication of displacement in depth. To reduce these effects, three-dimensional super-streamlines are alternatively rendered as a series of shaded variable-width tubes. Thinning the tubes at a percentage of their original calculated width reduces occlusion and shading improves the perception of vertical movement. Although the representation of super-streamlines as tubes do not directly resemble the illustrations we sought to emulate, the images may be enhanced with color-coding and texture mapped arrows to provide an equally effective abstract overview of the flow field.

4.2 Proposed Future Work

During the construction of these images, a large amount of the computation time is spent on calculating the principal streamline. Moreover, the time to compute the principal streamline increases with the size and resolution of the grid. Currently, principal streamline candidates are generated for each grid cell, the ideal candidate is stored and all other candidates are wastefully discarded. The performance of this process may be improved by employing an auxiliary grid and priority queue from which candidate principal streamlines are inserted and removed as needed throughout the course of the algorithm.

Animation of super-streamline illustrations would be an effective method for demonstrating the behavior of currents over a period of time. For example, currents often change as a product of seasonal conditions. Animating currents over two seasons or the period of one year could prove to be an effective way to demonstrate the relation of seasonal climate change to patterns in current flow. However, simply creating a series of images from a time ordered sequence of flow fields might not produce desirable results because currents might appear to flicker and jitter as the animation runs. A correlation between successive frames of the animation must be maintained to avoid these effects. Jobard and Lefer presented a method for animating a time series of their evenly spaced streamlines by maintaining a correlation between streamlines in successive frames [Jobard & Lefer, 2000]. Perhaps the results of their work could be used as a basis for correlating super-streamlines between successive frames.

Currently, the images produced show only relative differences in flow velocity, temperature and salinity throughout the flow field. For practical purposes, it would be useful to quantify data values represented by the illustrations. For example, arrows could be labeled with the current velocity and a color key could be used to translate the temperature and salinity values. Because oceanographers are often concerned with the transport of volumes of water, it might also be desirable to have a representation of density values.

The least effective aspect of our illustrations according to our design goals is the manner in which branching and merging of currents is represented. Currently, ribbons and tubes are simply extended a small amount beyond their calculated lengths in hopes that the objects will intersect in an aesthetically pleasing manner. However, this does not always produce the desired results. In a more sophisticated approach, super-streamlines could be stored in a graph or tree structure where a common node would be shared among intersecting super-streamlines. From this structure, a branch or merge may be detected and handled in the rendering process to provide a more desirable representation at intersections.

When computing and rendering super-streamlines with three-dimensional flows, the depth of the ocean was represented in highly exaggerated sigma-coordinates, i.e., the distance between sigma layers is equal to the horizontal data grid cell size. In reality, the depth of the ocean varies and is a small fraction of the distance covered in the latitude and longitude directions. This interpretation of the depth values was performed in order to 1) create a manageable space for algorithm development, and 2) to provide a depiction of depth in which vertical motion can be seen. However, this representation can produce misleading results. For example, flows occurring near the bottom of shallow water may appear as if they are at the same depth as flows occurring near the bottom of deep water, when in reality the difference in depth may be hundreds of meters.

One possible solution to this problem would be to use elliptical cross sections that are wide in the latitude-longitude plane and skinny in depth as a basis for calculating super-streamlines in a more realistic coordinate system. Another solution might be to scale and translate the coordinates of the super-streamlines of the current representation such that they conform to the contours of the ocean floor.

4.3 Conclusion

A method for automatic illustrations of ocean currents was developed with the intention that the images produced could be used in a variety of applications. Surface current illustrations could be a useful navigational guide for mariners or may serve as a reference to fishermen. The volume illustrations could be beneficial as a cross-reference for marine biologists studying migratory patterns in ocean life or could demonstrate or inspire an oceanographer's theory. Furthermore, the tube representations might serve as a handy data exploration tool in a geographic information system that incorporates flow data.

We have shown our results to a National Oceanic and Atmospheric Administration (NOAA) meteorologist and thus far obtained a positive response. NOAA is currently producing a wide range of climate and ocean flow model output on a regular basis and there is a need for ways of presenting the output of these models to various constituencies of users, including fishermen and recreational boaters. We have begun discussions to explore the possibility that versions of these diagrams may be used as a method for public dissemination of output from the Hybrid Coordinate Ocean Model (HYCOM) Gulf of Maine model.

REFERENCES

Abraham, R. H. and C. D. Shaw. (1992). *Dynamics – The Geometry of Behavior.* Addison-Wesley. Boston, MA.

Cabral, B. and L. Leedom (1993). "Imaging Vector Fields Using Line Integral Convolution." In *Proceedings of SIGGRAPH 93*, pp. 263-270. ACM SIGGRAPH, Anaheim , CA.

Chen, C., H. Hiu, and R. C. Beardsley (2003). "An Unstructured Grid, Finite-Volume, Three-Dimensional, Primitive Equations Ocean Model: Application to Coastal Ocean and Estuaries", In *Journal of Atmospheric and Oceanic Technology*, pp. 20,159-186. AMS, Boston, MA.

Evenden, G. (2000). "PROJ.4 – Cartographic Projections Library." Referenced on July 2004, http://www.remotesensing.org/proj/. Copyright © Frank Warmerdam.

Feynman, R. P., R. B. Leighton et al. (1964). *The Feynman Lectures on Physics*, Addison-Wesley, Reading, MA.

Fuhrmann, A. and E. Gröller (1998). "Real-Time Techniques For 3D Flow Visualization." In *Proceedings of the conference on Visualization '98*. pp. 305-312. IEEE, Research Triangle Park, NC.

Gad-el-Hak, M. (2000). *Flow Control: Passive, Active, and Reactive Flow Management*. Cambridge University Press , Cambridge, UK.

Gärtner, B. (1999). "Fast and Robust Smallest Enclosing Balls." In *Proceedings of the 7th Annual European Symposium of Algorithms*. pp. 325-338. Springer-Verlag, London, UK.

Gooch, A., B. Gooch, et al. (1998). "A Non-photorealistic Lighting Model for Automatic Technical Illustration." In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques."* pp. 447-452. ACM, New York, NY.

Hauser, H., R. Laramee and H. Doleisch (2002). "State of the Art Report 2002 in Flow Visualization." TR-VRVis-2002-03, Technical Report, VRVis Research Center, Vienna, Austria.

Hultquist, J. P. M. (1992). "Constructing Stream Surfaces in Steady 3-D Vector Fields." In *Proceedings of Visualization '92.* pp.171-178, IEEE, Boston, MA.

Jobard, B., G. Erlebacher and M. Y. Houssaini. (2002). “Lagrangian-Eulerian Advection of Noise and Dye Textures for Unsteady Flow Visualization.” In *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 211-222. IEEE.

Jobard, B. and W. Lefer (2000). “Unsteady Flow Visualization by Animating Evenly-Spaced Streamlines.” In *Computer Graphics Forum 19(3) (Proceedings of Eurographics 2000)*. Eurographics, Interlaken, Switzerland.

Jobard, B. and W. Lefer (1997). “Creating Evenly-Spaced Streamlines of Arbitrary Density.” In *Proceedings of Visualization in Scientific Computing ’97*. pp. 45-55. Eurographics, Boulogne sur Mer, France.

Kirby, R. M. , H. Marmanis, and D.H. Laidlaw (1999). “Visualizing Multivalued Data from 2D Incompressible Flows Using Concepts from Painting”, In *Proceedings of IEEE Visualization 1999*, pp. 333-340, IEEE, San Francisco, CA.

Laidlaw, D.H., R. M. Kirby, et al. (2001). “Quantitative Comparative Evaluation of 2D Vector Field Visualization Methods.” In *Proceedings of IEEE Visualization*. pp.143-150. IEEE, San Diego, CA.

Löffelmann, H., L. Mroz, and E. Gröller (1997). “Hierarchical Streamarrows for the Visualization of Dynamical Systems.” In *Proceedings of Visualization in Scientific Computing ’97*. pp. 155-164. Eurographics, Boulogne sur Mer, France.

Lorensen, W. E. and H. E. Cline (1987). “Marching Cubes: A High Resolution 3D Surface Construction Algorithm.” In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques.* Vol. 21, No. 4. pp. 163-169. ACM, New York, NY.

Lu, A., C. J. Morris, et al. (2002). “Non-photorealistic Volume Rendering Using Stippling Techniques.” In *Proceedings of the conference on Visualization ’02.* pp. 211-218. IEEE. Boston, MA.

Press, W. H., S. A. Teukolsky, et al. (1992). *Numerical Recipes in C: The Art of Scientific Computing, 2nd ed.* Cambridge University Press, Cambridge, UK.

Scheuermann, G., T. Bobach, et al. (2001). “A tetrahedral-based stream surface algorithm.” In *Proceedings of IEEE Visualization ’01.* pp.151-158. IEEE. San Diego, CA.

Scheuermann, G., H. Hagen, et al. (1997). “Visualization of higher order singularities in vector fields.” In *Proceedings of IEEE Visualization ’97.* pp. 67-74. IEEE. Phoenix, AZ.

Schroeder, W., K. Martin and B. Lorensen (2004). *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 3rd ed.* Prentice Hall, New Jersey, 2004.

Schulz, M., F. Recks et al. (1999). "Interactive visualization of fluid dynamics simulations in locally refined Cartesian grids." In *Proceedings of IEEE Visualization '99.* pp. 413-416. IEEE. San Francisco, CA.

Shewchuk, J. R. (1996). "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator." In *First Workshop on Applied Computational Geometry.* pp.124-133. ACM, Philadelphia, PA.

Tufte, E. R. (2001). *The Visual Display of Quantitative Information, 2nd ed.* Graphics Press, Cheshire, CT.

Turk, G. and D. Banks (1996). "Image-Guided Streamline Placement." In *Proceedings of SIGGRAPH 96.* pp. 453-460. ACM SIGGRIAPH, New Orleans, LA.

van Wijk, J. J. (1991). "Spot Noise – Texture Synthesis for Data Visualization." In *Computer Graphics (SIGGRAPH '91 Conference Proceedings),* pp. 263-272. ACM SIGGRAPH, Las Vegas, NV.

van Wijk, J. J. (1993). "Implicit Stream Surfaces." In *Proceedings of the 4th conference on Visualization '93.* pp. 245-252, IEEE, San Jose, CA, 1993.

M. Ward (1997). "Flow Visualization Techniques." Referenced on January 2005, http://www.cs.wpi.edu/~matt/courses/cs563/talks/flowvis/flowvis.html. Worcester Polytechnic Institute, Worcester, MA.

C. Ware (2004). *Information Visualization: Perception for Design, 2nd ed.* Morgan Kaufmann. San Francisco, CA.

Winkenbach, G. and D. H. Salesin (1994). "Computer-Generated Pen-and-Ink Illustration." In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques.* pp. 91-100. ACM, New York, NY.

APPENDIX A

FIGURE CREDITS

The following figures were reprinted with permission:

1	"Undersea Landscapes of the Gulf of Maine." Courtesy of Gulf of Maine Council on the Marine Environment.
3	Jobard et al. (2002). Copyright © 2002. IEEE.
4	Design by Jack Cook. Courtesy of Woods Hole Oceanographic Institute.
5	Extracted from Dinter (2001): "Biogeography of the OSPAR Maritime Area." German Federal agency for Nature Conservation. Courtesy of MAR-ECO.
6	Design by Philippe Rekacewicz. Courtesy of United Nations Environment Programme / GRID-Arendal
7	http://www.efluids.com/efluids/gallery/gallery_pages/da_vinci_page.htm Courtesy of Dr. Mohamed Gad-el-Hak.
8,10,16	Laidlaw et al. (2001). Copyright © 2001. IEEE.
9	Schultz (1999). Copyright © 1999. IEEE.
15	Scheuermann et al. (2001). Copyright © 2001. IEEE.
17	Scheuermann et al. (1997). Copyright © 1997. IEEE.
18	Winkenbach and Salesin (1994). Copyright © 1994. Association for Computing Machinery, Inc.
19	Gooch et al. (1998). Copyright © 1998. Association for Computing Machinery, Inc.
20	Lu et al. (2002). Copyright © 2002. IEEE.

21	Kirby et al. (1999). Copyright © 1999. IEEE.
22	Turk and Banks (1996). Copyright © 1996. Association for Computing Machinery, Inc.
23,24	Jobard and Lefer (1997). Copyright © 1997. European Association for Computer Graphics.
26	Löffelmann et al. (1997). Copyright © 1997. European Association for Computer Graphics.