VISUALIZATION OF SURFACES AND 3D VECTOR FIELDS

Wentong Li, M. Sc.

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2002

APPROVED:

Robert J. Renka, Major Professor
Tom Jacob, Committee Member
Karl Steiner, Committee Member
Robert Brazile, Chair of Graduate Studies in the Department of Computer Science
Krishna Kavi, Chair of the Department of Computer Science
C. Neal Tate, Dean of the Robert B. Toulouse School of Graduate Studies

Li, Wentong, Visualization of Surfaces and 3D Vector Fields. Master of Science (Computer Science), August 2002, 43 pp., 3 tables, 10 illustrations, 9 references.

Visualization of trivariate functions and vector fields with three components in scientific computation is still a hard problem in compute graphic area. People build their own visualization packages for their special purposes. And there exist some general-purpose packages (MatLab, Vis5D), but they all require extensive user experience on setting all the parameters in order to generate images. We present a simple package to produce simplified but productive images of 3-D vector fields. We used this method to render the magnetic field and current as solutions of the Ginzburg-Landau equations on a 3-D domain.

ACKNOWLEDGMENTS

I thank the University of North Texas for financial support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

1. INTRODUCTION

A three-dimensional(3D) vector field in R^3 is a mapping from R^3 to R^3. Visualization of a 3D object requires projecting the object to a 2D domain before we can visualize it by the monitors or the plotters. Visualization of 3D vector fields is a hard problem. A global view of a 3D vector field means an image that contains all the information of a 3D vector field data set. A local view of a 3D vector field means an image that contains only the information of a subset of a 3D vector field data set. It is nearly impossible to provide a global view of a 3D vector field. Scientists are still developing methods to display 3D vector fields.

In this thesis, we present a simple method that can give both the local view and the global view of 3D vector fields. The local view of a 3D vector field can be provided by displaying the cross-sections within the domain. An efficient way of displaying the cross-sections is to restrict the cross-sections to the planes with the coordinate axes as normals. We add the color-filled contour

plots to the surfaces to represent the magnitudes of the vectors in the planes. We also add the arrow plots to the surfaces to represent the directions of the vectors in the planes. The global view of a 3D vector field is given by animating the surfaces moving along the normal directions and by adding the streamlines into the 3D vector fields. The animation of the surfaces will give global magnitude information of the 3D vector fields. The streamlines can illustrate the global directional information of a vector field.

The data sets that we use in this method are the 3D vector data sets that are associated with the regular grid points in a cubic domain.

We begin the thesis by giving an overview of the background of visualization in the scientific computation. Then, we will review the OpenGL library that is used in the implementation of the software. After that, we will introduce the design, the data structures and some algorithms in the implementation. Finally, we will give some visualization results.

2. BACKGROUND

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube (CRT) screens as early as the introduction of computer. Scientific visualization date back to the start of computer graphics, and it is a major sub-field in computer graphics.

Visualization can be defined as a method of extracting meaningful information from complex data sets through the use of interaction graphics and image. Visualization in scientific computing deals with techniques that allow scientists and engineers to extract knowledge and to formulate their results from complex simulations and computations. "As a tool for applying computers to science, visualization offers a way to see the unseen. As a technology, visualization in scientific computing promises radical improvements in the human/computer interface and may make human-in-the-loop problems approachable"[1].

2.1 Vector graphics vs. raster graphics

The display of graphics in the sixties and seventies was based on vector drawing devices. A vector drawing system is one that stores instances of graphic primitives as parametric types and data values. It performs an object-

based approach to scene representation, manipulation, and display. A geometric representation of the objects comprising the scene is stored in a display-list. Screen refreshing is accomplished by redrawing the vectors comprising the objects in the display list. The major advantages of vector graphics are its ability to perform object-related operations on the display list and its ability to draw vectors continuously, thus exhibiting no aliasing. This technology, however, offers calligraphic drawing only while the interior shaded areas are extremely hard to render. The vector based display devices were also very expensive at that time.

With the development of CRT-based raster graphic devices, display devices became cheaper in the late seventies, and since then, raster graphic devices have dominated the graphic devices market. The algorithms that are used in vector graphic have been modified to be used on the raster devices. Raster graphic devices use a 2D frame-buffer, a raster, of pixels for scene representation and a pixel-based rendering for coloring those pixels that correspond to the discrete representation of the geometric objects. A video controller performs screen refreshing, which repeatedly displays the frame-buffer onto the screen [2].

Table 1 compares vector graphics with raster graphics.

Capability	Vector-Graphics	Raster-graphics
1. Rendering and Screen-refresh	Rendering is embedded in screen refreshing	Scan-conversion decoupled from screen-refreshing
2. Refreshing performance	Sensitive to scene complexity	Insensitive to scene complexity
3. Memory and processing requirement	Depends on the scene and object complexity	Constant often very large
4. Screen space Aliasing	None	Yes
5. Transformation	Performs on objects	Perform on the pixels
6. Rendering of interior	No, boundary only	Colored, shaded and textured surface
7. For scientific computing data sets	Not enough	Very good
8. Measurements(e.g., distance, area)	Analytical, but often complex	Approximation, simple

Table 1. A comparison between vector graphics and raster graphics

From Table 1, we can see the major advantage of raster devices is that they separate the image generation process

from screen refreshing, thus making the refreshing of the screen independent from the complexity of the scene. With the advances in hardware development and the development of antialiasing methods, raster graphic devices have replaced vector graphic devices as the primary technology for computer graphics. The CRT-based raster graphic devices are used by nearly all PCs and workstations today.

2.2 Scientific visualization

Scientists, engineers, and medical workers often need to analyze large amounts of information or to study the behavior of certain processes. Generating and processing graphical representation for scientific, engineering and medical data sets are generally referred to as scientific visualization. The data sets can be gathered from instruments or the results of simulations. Nowadays, the data sets are becoming larger and larger. The purpose of the visualization in scientific computation is to help the user get a better understanding of the meaning of the data.

The data sets of scalar fields and vector fields are very common in the scientific computation. Scalar fields are data sets that have only magnitudes over the domain (time or

spatial). Vector fields are data sets that have both magnitude and directional information over the domain.

Generally, we can divide the problem into two categories: scalar field visualization and vector field visualization. According to the dimension of the domain, we can divide the problem into 0D, 1D, 2D, 3D, and high dimensions. 0D and 1D problems are trivial because we already have 2D domains to display the data sets -- screens or plotters. Most problems in the scientific computation area are 2D or 3D.

2D scalar field visualization

A two-dimensional scalar field is a function of two variables:

$$z = f(x, y) \qquad x, y \in N.$$

The display devices(screens or plotters) have the same domain dimension as 2D scalar fields, so we have two methods to represent 2D vector fields.

The first method is the color-mapping (color-coding) method. We map discrete or continuous colors to the scalar fields according to the magnitude of the scalar. Figure 1 shows an example of color mapping.

The second method uses a 3D surface to represent a 2D scalar data set. We interpolate the data values to display a

smooth surface, and we can use the light reflection of the surfaces to distinguish the scalar magnitude on the surface. Figure 2 shows an example of the surface representation of 2D scalar fields.

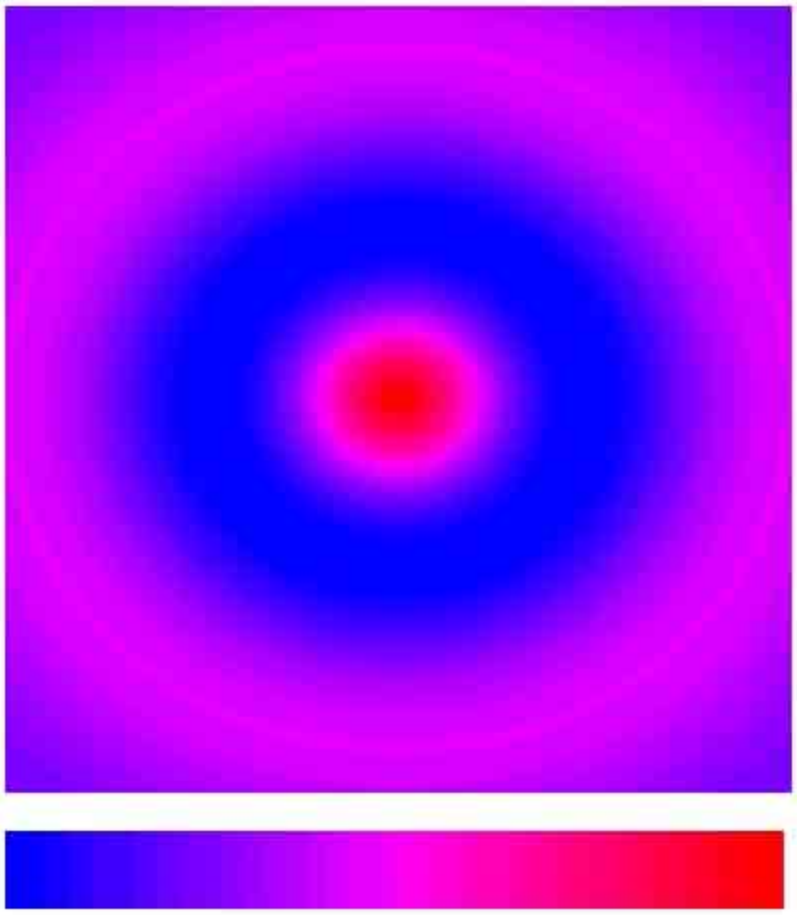


Figure 1. The color-coded image of function
$f(x,y) = \exp[-0.04 * sqrt((80x-40)^2 + (90y-45)^2)] * \cos[0.15 * sqrt((80x-40)^2 + (90y-45)^2)]$
in domain D=[0,1]*[0,1]

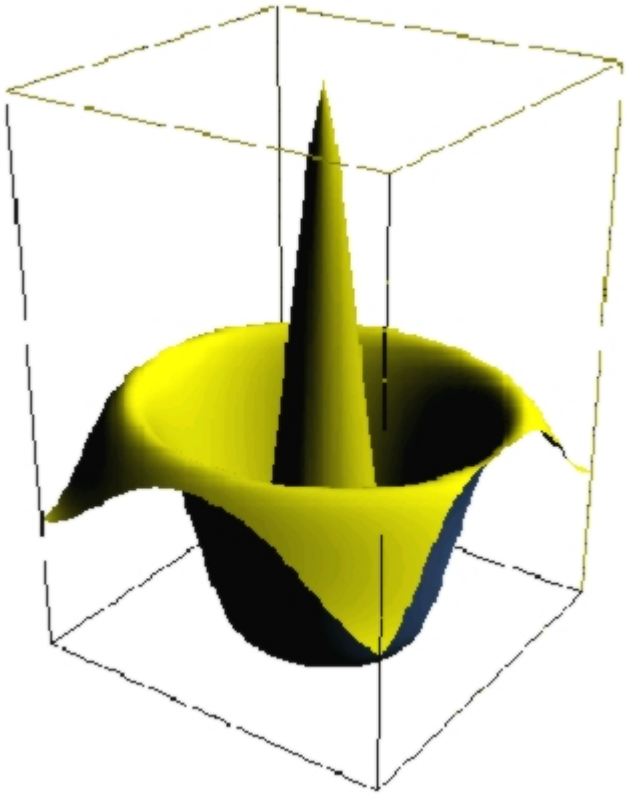


Figure 2. The surface image of function

$$f(x,y) = \exp[-0.04 * sqrt((80x-40)^2 + (90y-45)^2)] * \cos[0.15 * sqrt((80x-40)^2 + (90y-45)^2)]$$

in domain D=[0,1]*[0,1]

2D vector field visualization

A vector field in R^2 is a mapping from R^2 to R^2; i.e., an ordered pair of functions (u(x,y), v(x,y)):

$$m: R^2 \rightarrow R^2$$

$$m(x,y) = \begin{pmatrix} u(x,y) \\ v(x,y) \end{pmatrix}$$

The vector field value at a point (x,y) is represented by an arrow from (x,y) to(x+s*u(x,y),y+s*v(x,y)), where s is a scale factor chosen so that we can display the vectors reasonably within the domain. We can also combine the color-mapping method with the arrow plot method to represent 2D vector fields.

In the above 2D scalar field and vector field visualization methods, we can represent all the information of a data set in one picture.

3D scalar fields visualization

A three-dimensional scalar field is a function of three variables; i.e.,

$$w = f(x, y, z) \qquad x, y, z \in N.$$

In this case, the data set we are going to represent has a higher dimension than the domain that is used to represent it, so there must be some strategy to reduce the dimension of the data set or to create a new mapping method to force the 3D scalar data set to be represented on the 2D screen or plotter.

The following methods have been used to represent a 3D scalar field.

Method a) represents the partial information of the data sets by rendering the cross-section surfaces in the domain.

Method b) renders one or more isosurfaces in the domain to give a global view of the data set.

Method c) uses the volumetric rendering method. Volume rendering is a technique for directly displaying a sampled 3D scalar field without first fitting geometric primitives to the samples. There are many algorithms for volume rendering [3][4].

There are advantages and disadvantages of these methods. Method a) can give a precise representation of a cross section, but it can't give a global view of the data set. Method b) can give a kind of global view of the whole data set, but it take much computation to construct the surface and it also can only give partial information about the data set. Method c) can give a global view of the data set, but for a single point in the domain it does not give an accurate representation. It works very well for the data set generated by the CT or MRI, but for a computational data set, sometimes it does not provide meaningful information.

3D vector field visualization

A vector field in R^3 is a mapping from R^3 to R^3; i.e., an ordered triple of functions (u(x,y,z), v(x,y,z), w(x,y,z)):

$$m: R^3 \rightarrow R^3$$

$$m(x,y,z) = \begin{pmatrix} u(x,y,z) \\ v(x,y,z) \\ w(x,y,z) \end{pmatrix}$$

The visualization of 3D vector fields is very difficult. There is no general method used in the display of 3D vector fields. There are some methods used to represent the partial information of 3D vector fields: using arrow plots to represent the directional information of the cross sections in the vector fields or using texture mapping along with volume rendering to provide both the directional and magnitude information of the whole data set. Adding streamline, streamribbon, and streamtube to the 3D vector fields can help give the global directional information.

However, 3D vector fields are too complicated such that one technique works well with some data sets but not others.

2.3 Human perception and the visibility of information in the data

The understanding of human perception could help improve the visibility of information in the displayed data. For example, using a suitable color scale to depict data values and an appropriate brightness contrast between adjacent regions in an image could improve the visibility of information embedded in 2D and 3D data.

In the color-coding visualization system, color is a perceived sensation rather than a wavelength. Color perception is a complicated process for which there is no complete theory. However, there are some guidelines that could help the user improve the visibility of information embedded in the displayed data. One example is the use of pseudo-color to represent the magnitudes of the data points.

In a pseudo-color representation, each value or a range of values is associated with a color. The rainbow color scale, in which the assignment of a color to a data value or to a range of value is determined by the position of the color in the visible spectrum, is the most commonly used scale. A color scale based on the brightness contrast is recommended rather than on the hue contrast [5]. In this scheme, the higher the data value the brighter is the color (where the brightness is determined by the perceived

brightness rather than the physical value of the brightness). Using a brightness contrast-based color scale will help in discriminating not only data values but also the overall shape of the object.

When we represent data using colors, we should be aware that the bright objects on the dark background look bigger than the same objects depicted using darker color on bright background. This phenomenon is called irradiation [6]. However, under most circumstances the perceived size of an object is not very important. We could employ this phenomenon to make a small object look larger by using a bright color on a dark background.

The perception of the distance of an object to the user is also affected by the use of color scheme. For example, a rectangular button on the window with bottom- and right-black edges and upper- and left-white edges appears closer to the user. And the same button with bottom- and right-white edges and upper- and left-black edges looks far from the user. People use this to implement the press and release of buttons in a window system.

2.4 Overview of OpenGL

We use OpenGL as our graphic programming interface in implementing the package we present.

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. The OpenGL programming interface (API) was first developed at Silicon Graphics and has become the most widely used and supported 2D and 3D graphics standard. The OpenGL libraries contain about 150 distinct commands that programmer can use to specify the objects and operations needed to produce interactive three-dimensional graphic applications.

OpenGL provides a layer of abstraction between graphics hardware and an application program. OpenGL is an open interface that is designed to be streamlined and hardware independent, and which is implemented by many companies and organizations. There are commercial ports of OpenGL to Linux, but in this program we use a high-quality public domain OpenGL-like implementation called Mesa. Mesa cannot be called OpenGL because it is not licensed from Silicon Graphics, but it is an effective implementation of the OpenGL API in Linux.

In OpenGL, the objects are built up with a set of geometric primitives – points, lines, and polygons. The

OpenGL routines render the primitives onto a frame buffer and generate high-quality color images of 3D objects.

OpenGL is a state machine. The user puts it into various states or modes that then remain in effect until they are changed. Figure 3 shows the rendering pipeline of OpenGL.

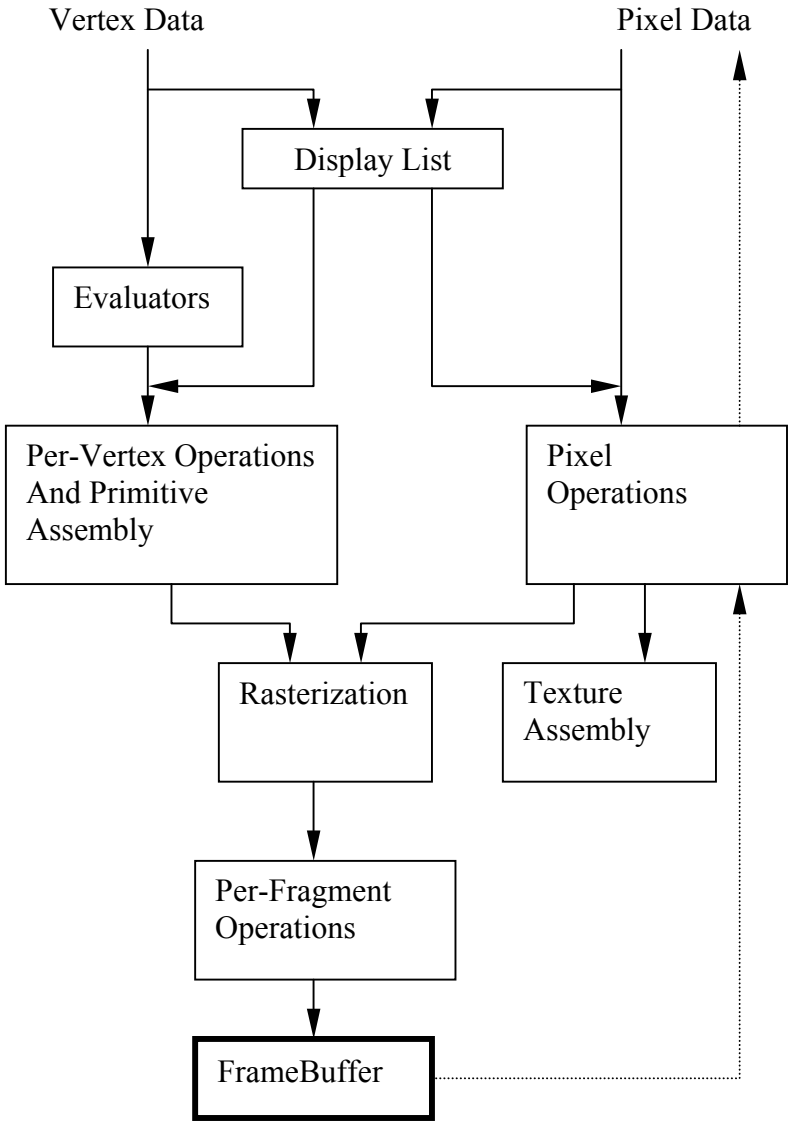


Figure 3. Rendering pipeline of OpenGL [7]

The client-server model is used for interpretation of OpenGL commands. This is just an abstract model; it does not demand that OpenGL be implemented as distinct client and server processes. A client-server approach means that the boundary between a program and the OpenGL implementation is well defined. This allows OpenGL to operate over a wire protocol, the way the X protocol operates, but OpenGL does not require that OpenGL rendering take place in a separate process. Using a wire protocol means that all OpenGL operations can be encoded in a stream of bytes that can be sent across a network. In case of the X window system, an X program using OpenGL can run on a remote computer while the results are displayed on a local workstation.

3. IMPLEMENTATION

3.1 Overall Design

The system we presented was designed to display the 3D vector field data sets that are associated with regular grids in the 3D rectangular domains.

It is very important for a scientific visualization system to have the ability to interact with the users. We add the interactive functions such as zoom, animation, rotation, and etc. into the system. These functions can help the users to get a comprehensive view of the data sets and to get a better understanding of the data. We enable the users to interact with the system by using the mouse and the keyboard. Table 2 shows the functions associated with the mouse and the keyboard.

Device Name	Function
Keyboard	Add name, functions associated with keys
Mouse	Rotation, Zoom , Add or delete streamlines, show the menu, choose menu item.

Table 2. Functions that associated with the keyboard and mouse

Table 3 shows some important functions that are included in the software.

	Major functions in the system	Control keys	Mouse Control
1	Zoom in	PgUp	middle mouse button
2	Zoom out	PgDn	shifted middle mouse button
3	Set current animation direction to x axis	F1	
4	Set current animation direction to y axis	F2	
5	Set current animation direction to z axis	F3	
6	Move the surface forward	>	
7	Move the surface backward	<	
8	Compute bench mark	B	
9	Toggle color-filled contour plot	c	
10	Toggle color-coding of arrows	C	
11	Increase the streamline step-size	D	
12	Decrease the streamline step-size	d	
13	Toggle display of arrow grid	g	
14	Toggle display of grid lines	G	
15	Toggle antialiasing	j	
16	Increase line width	K	
17	Decrease line width	k	
18	Increase the vector length cut off	M	
19	Decrease the vector length cut off	m	
20	Increase the signed distance to the center	P	

21	Decrease the signed distance to the center	p	
22	Query current state	q	
23	Dump current window into a postscript file	w	
24	Increase arrow density in x direction	X	
25	Decrease arrow density in x direction	x	
26	Increase arrow density in y direction	Y	
27	Decrease arrow density in y direction	y	
28	Increase arrow density in z direction	Z	
29	Decrease arrow density in z direction	z	
30	Add streamlines to the image	+	Use mouse button
31	Delete streamlines from the image	-	Use mouse button
32	Remove all streamlines	R	
33	Restore defaults	r	
32	Rotation		Left button down and mouse movement
33	Add title to the image	t	
34	Toggle arrowhead fill mode	f	
35	Terminate the program	'Esc'	

Table 3. Software main Functionalities

In this system, most functions are accessible by both the keyboard and the mouse except the rotation, which can only be controlled with the mouse.

Because the OpenGL library routines will delegate the clipping, view port mapping, and projection, we only need specify some parameters that we used by these processes.

3.2 Input Data format

The input data of the program is in the following format:

```
nx ny nz   -- numbers of the grid points in each
direction
xmin xmax  -- range of x values (xmin < xmax)
ymin yamx  -- range of y values (ymin < ymax)
zmin zmax  -- range of z values (zmin < zmax)
vx vy vz   -- components of first vector
vx vy vz   -- components of second vector
. . .
vx vy vz   -- components of last vector.
```

For l = 0 to nx*ny*nz-1, the l-th vector (vx(l), vy(l), vz(l)) is the value at grid point (x(i),y(j),z(k)) for x(i) = xmin + i*(xmax-xmin)/(nx-1), y(j) = ymin + j*(ymax-ymin)/(ny-1), z(k) = zmin + k*(zmax-zmin)/(nz-1), where l = k*nx*ny + j*nx + i, for i = 0 to nx-1, j = 0 to ny-1, and k = 0 to nz-1.

3.3 Data structure

The data structure involved in this system is made up of several one-dimensional arrays.

a) Data Array

The array "***vectors***" is an array used to store the vector data set correspondent with its grid point in the domain. This array is of length 4*nx*ny*nz (nx, ny and nz are the grids points in the x, y and z direction) containing the u, v, and w components of the vector and is followed by the vector magnitude. According to the format of the input data, the u, v, and w components of l-th vector are stored at index 4*l, 4*l+1, 4*l+2 of the array, and the magnitude of the l-th vector stored at index 4*l+3. Therefore, the index of the i-th element of the array is a component of (i/4)-th vector and the position of the vector data in the domain can be computed as:

```
x = xmin + ((i/4)%nx)*(xmax-xmin)/(nx-1)
y = ymin + ((i/4)/nx)*(ymax-ymin)/(ny-1)
z = zmin + ((i/4)/(nx*ny))*(zmax-zmin)/(nz-1)
```

where nx, ny, nz are the value of grids points in the x, y, z direction.

The array "***indc***" is an array of size nx*ny*nz that stores the contour level of the grid points.

b) Triangle list

There are three arrays used to save the triangle lists of the surfaces parallel to x-y, y-z, and z-x planes. The size of the arrays are 6*(nx-1)*(ny-1), 6*(ny-1)*(nz-1), and 6*(nz-1)*(nx-1). The triangle lists are stored before the rendering process begins. According to the input data format, we can index the grid points at the axis as figure 4.

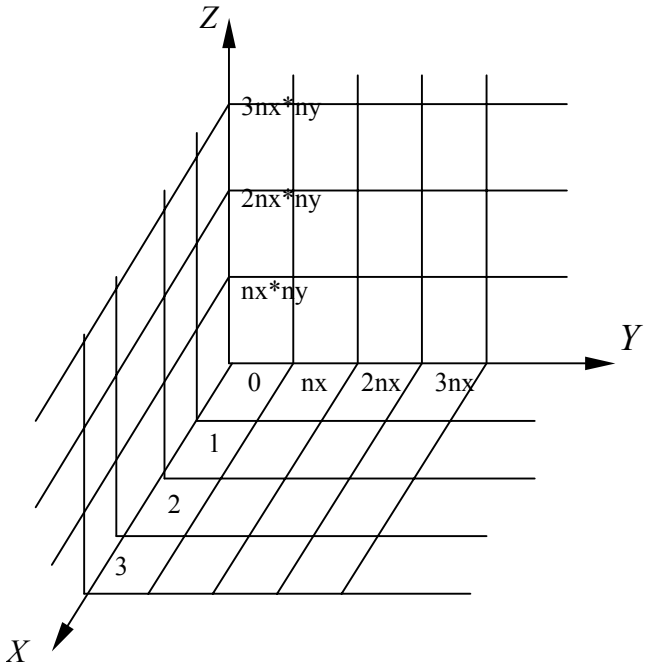


Figure 4. Indices of Grid points at the axis

From Figure 4, we can see that the triangle lists of the surfaces parallel to x-y plane can be easily computed by adding the index of the grid point at the z-axis of the corresponding surface to the triangle list of x-y plane. The

triangle lists of the surfaces that parallel the y-z or x-z plane can also be computed similarly.

The triangle list of the surface in the xy-plane can be computed as follows:

```
k=0;
for( i= 0 to ngy-1 ){
   for( j = 0 to ngx-1 ){
        index[k++]= ngx*i + j;
        index[k++]= ngx*i + j + 1;
        index[k++]= ngx*(i+1) + j + 1;
        index[k++]= ngx*i + j;
        index[k++]= ngx*(i+1) + j + 1;
        index[k++]= ngx*(i+1) + j;
   }
}
```

The triangle list of the surface in the yz-plane can be computed as follows:

```
k=0;
for( i= 0 to ngz-1 ){
   m = i*ngxy;
   for( j = 0 to ngy-1 ){
        index[k++]= m;
        index[k++]= m + ngx;
```

```
            index[k++]= m + ngx + ngxy;
            index[k++]= m;
            index[k++]= m + ngx + ngxy;
            index[k++]= m + ngxy;
            m = m + ngx;
      }
   }
```

The triangle list of the surface in the xz-plane can be computed as follows:

```
   k=0;
   for( i= 0 to ngz-1 ){
      m = i*ngxy;
      for( j = 0 to ngx-1 ){
            index[k++]= m;
            index[k++]= m + 1;
            index[k++]= m + 1 + ngxy;
            index[k++]= m;
            index[k++]= m + 1 + ngxy;
            index[k++]= m + ngxy;
            m = m + 1;
      }
   }
```

C) Color table

A pre-stored RGB color table is used to specify the different contour colors. The color table starts with only a pure blue component, then we increase the red component. After that, we decrease the blue component and the coloe table ends at a pure red component. There is no green component used in this color table. The color bar is the same as in Figure 1.

3.4 Techniques and Algorithms

a) Data Interpolation

There are three types of data interpolation involved in the system -- linear interpolation, bilinear interpolation, and trilinear interpolation of data. The linear interpolation is used in making the color-filled contour plot of the surfaces. The bilinear interpolation is used in adding the arrow plots into the surfaces, and the trilinear interpolation is used in calculating the streamlines.

b) Color-filled contour plot

In this program, we provide a color-filled contour plot by filling each of the triangles in the surfaces. Each grid of the surface is divided into two triangles. We fill the

triangles by linear interpolating the magnitude of the vectors at the vertices of the triangles, and then, fill the polygons with colors according to then contour value. For the surface that parallels the x-y plane there are 2(nx-1)(ny-1) triangles that need to be filled, for the surface parallel to y-z plane, 2(ny-1)(nz-1) triangles, and for the surface parallel to zx-plane, 2(nz-1)(nx-1) triangles.

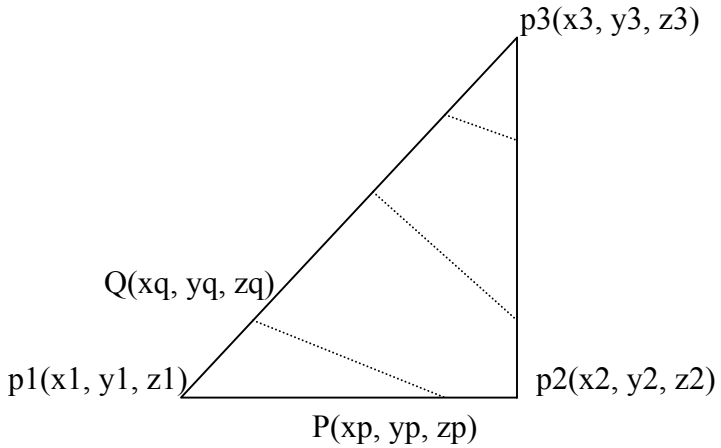


Figure 5. Partition of a triangle

There are some pre-defined arrays that are used in this algorithm:

Colors[NC][3] – Array of size NC*3 that stores the RGB components of each contour level.

Contours[NC] – Array of length NC that stores an increasing sequence of vector magnitude values defining the upper limits of the contour levels.

By observing Figure 5, we can see using two lines to partition a triangle linearly into polygons so that the

largest number of vertices of the polygons is five. Therefore, we allocate an array **vertex** of size 5 by 3 as working space. The algorithm for color-filling a contour surface is as follows:

```
CfillTriangle(int i1, int i2, int i3)
  // i1, i2, i3 are the indices of the triangle
  permute(i1, i2, i3) so that indc[i1]<=indc[i2] &&
                              indc[i1]<= indc[i3];
  ic1 = indc[i1], ic2 = indc[i2], ic3 = indc[i3];
  m1 = vector[4*i1+3];
  m2 = vector[4*i2+3];
  m3 = vector[4*i3+3];
  compute the coordinates(x1,y1,z1) of index i1 vertex;
  compute the coordinates(x2,y2,z2) of index i2 vertex;
  compute the coordinates(x3,y3,z3) of index i3 vertex;
  set n = 0;
  //n is used to count the vertices number of a polygon.
  vertex[n++] = (x1,y1,z1);
  for(ic = ic1; ic < ic2 ; ++ic){
    m = contour[ic];
    s = (m-m1)/(m2-m1);
    xp = x1 + s*(x2 – x1);
```

```
    yp = y1 + s*(y2 – y1);
    zp = z1 + s*(z2 – z1);
    if(ic < ic3){ // q is at p1-p3 (see figure 5.)
       s = (m-m1)/(m3-m1);
       xq = x1 + s*(x3 – x1);
       yq = y1 + s*(y3 – y1);
       zq = z1 + s*(z3 – z1);
    } else {// q is at p2-p3 (see figure 5.)
       s = (m-m3)/(m2-m3);
       xq = x3 + s*(x2 – x1);
       yq = y3 + s*(y2 – y1);
       zq = z3 + s*(z2 – z1);
    }
    vertex[n++] = (xp,yp,zp);
    vertex[n++] = (xq,yq,zq);
    if(ic == ic3) vertex[n++] = (x3,y3,z3);
    fill_Polygon(vertex , n, ic);
    n = 2;
    vertex[0] = (xq,yq,zq);
    vertex[1] = (xp,yp,zp);
}
vertex[n++] = (x2,y2,z2);
if(ic3 >= ic2){
```

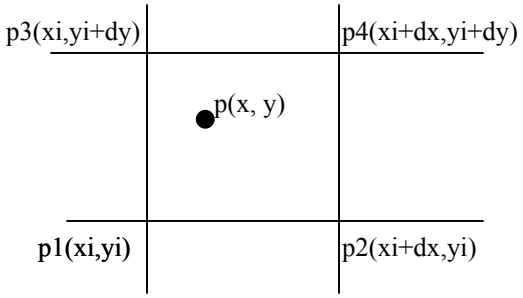
```
      for( ic=ic2 ; ic<ic3 ; ic++ ){
         m = contour[ic];
         s = (m-m2)/(m3-m2);
         xp = x2 + s*(x3 - x2);
         yp = y2 + s*(y3 - y2);
         zp = z2 + s*(z3 - z2);
         s = (m-m1)/(m3-m1);
         xq = x1 + s*(x3 - x1);
         yq = y1 + s*(y3 - y1);
         zq = z1 + s*(z3 - z1);
         vertex[n++] = (xp,yp,zp);
         vertex[n++] = (xq,yq,zq);
         fill_Polygon(vertex , n, ic);
         n = 2;
         vertex[0] = (xq,yq,zq);
         vertex[1] = (xp,yp,zp);
      }
      vertex[n++] = (x3,y3,z3);
      ic = ic3;
   }
   fill_Polygon(vertex , n, ic);
end.
```

This algorithm fills a triangle each time it is called. In order to render a picture, 2*(ngx-1)*(ngy-1)+2*(ngy-1)*(ngz-1)+2*(ngz-1)*(ngx-1) triangles need to be filled.

c) Arrow plot

We add arrow plots into the current active surfaces in the x, y, and z directions. The tails of the arrows are at the surfaces. The components of the arrows are computed by piecewise-bilinear interpolation of the four corner grid points that contain the tail of the arrow. The number of arrows is at least 2 in each direction.

The following is an example of piecewise-bilinear interpolation of the data in the x-y plane.



$$\tilde{V}_p = \begin{pmatrix} \tilde{V}_{p1} & \tilde{V}_{p2} & \tilde{V}_{p3} & \tilde{V}_{p4} \end{pmatrix} \begin{pmatrix} b1 \\ b2 \\ b3 \\ b4 \end{pmatrix}$$

```
b1 = (xi+dx-x)*(yi+dy-y)/(dx*dy);
b2 = (xi-x)*(yi+dy-y)/(dx*dy);
b3 = (xi+dx-x)*(yi-y)/(dx*dy);
b4 = (xi-x)*(yi-y)/(dx*dy);
```

The psuedo-code to add the arrow plots to the surface parallel to xy-plane is as follows:

```
DrawField(double hx, double hy)
//hx, hy are the distances between arrow tails
  in x, y direction
  int ngx, ngy; // number of grids in x,y direction
   x = xmin, y = ymin; //lower left corner of the domain
   xg = dx, yg = dy;  // dx, dy are the data grid width
   int jcnt = 0; // counter in y direction
   for(j=0; j<nay; j++){
      x=xmin;
```

```
      icnt = 0;
      for(i=0; i<nax; i++){
         Compute the vecctor at(x,y,z);
         drawArrow(x,y,z,vx,vy,vz);
         x = x+hx;
         advance to next grid in x direction;
      }
      hg = hxg ;
      y = y+hy;
      advance to next grid in y direction;
     }
end.
```

d) Streamline

A streamline is the path of a massless particle that is released in a steady flow. The plotting of the particle paths produces a streamline picture, which is of both qualitative and quantitative value to an engineer.

We compute a sequence of points that define a streamline beginning at (x0,y0,z0) by using Euler's method.
The sequence of points u = (ux, uy ,uz) is computed by integrating the system of ODE's:

u'(t) = V(u(t))/(|V(u(t))|)= (vx, vy, vz)/sqrt(vx^2+vy^2+vz^2), where u(0) = (x0, y0, z0); vx(ux, uy, uz), vy(ux, uy, uz), and vz(ux, uy, uz) are obtained by piecewise trilinear interpolation of the gridpoint vectors. Therefore,

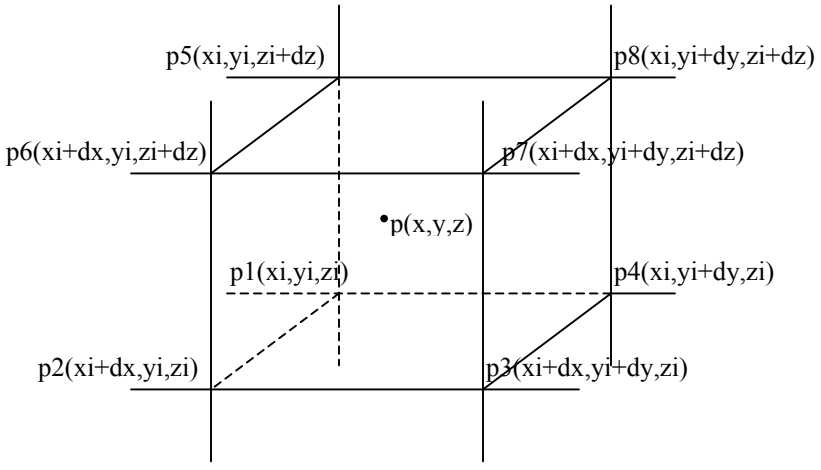
$$u(i) = u(i-1) + \frac{V(u(i-1))}{|V(u(i-1))|} \bullet dt$$

i>1 and dt is the step size.

In this program, the streamline is terminated when it intersects itself (enters a cell previously encountered), hits the boundary of the domain pointing outward, encounters a point where the magnitude of the vector is less than a threshold value, or reaches the maximum number of points in a streamline.

The starting point of the streamline is obtained from the mouse position in the domain.

The following is an example of the piecewise trilinear-interpolation of the data in a cell.



$$\tilde{V}_p = \begin{pmatrix} \tilde{V}_{p1} & \tilde{V}_{p2} & \tilde{V}_{p3} & \tilde{V}_{p4} & \tilde{V}_{p5} & \tilde{V}_{p6} & \tilde{V}_{p7} & \tilde{V}_{p8} \end{pmatrix} \begin{pmatrix} b1 \\ b2 \\ b3 \\ b4 \\ b5 \\ b6 \\ b7 \\ b8 \end{pmatrix}$$

```
b1 = (xi+dx-x)*(yi+dy-y)*(zi+dz-z)/(dx*dy*dz);

b2 = (x-xi)*(yi+dy-y)*(zi+dz-z)/(dx*dy*dz);

b3 = (x-xi)*(y-yi)(zi+dz-z)/(dx*dy*dz);

b4 = (xi+dx-x)*(y-yi)*(zi+dz-z)/(dx*dy*dz);

b5 = (xi+dx-x)*(yi+dy-y)*(z-zi)/(dx*dy*dz);

b6 = (x-xi)*(yi+dy-y)*(z-zi)/(dx*dy*dz);

b7 = (x-xi)*(y-yi)(z-zi)/(dx*dy*dz);

b8 = (xi+dx-x)*(y-yi)*(z-zi)/(dx*dy*dz);
```

The psuedo-code of computing a streamline is as follows:

```
ComputeStreamline(double x0, double y0, double z0)
//x0, y0 and z0 are the coordinates of the starting point
  append the (x0, y0, z0) into the streamline;
  copy (x0 , y0 , z0) to (x1, y1, z1);
  compute the cell contains (x1, y1, z1);


 n = 1;
 while( n < max value){
```

```
      compute the V at (x1,y1,z1) by trilinear interpolation
      of the grid point;
      if(|V| < vmin) return;
      x2 = x1 + dt*vx , y2 = y1 + dt*vy, z2 = z1 + dt*vz;
      compute the cell contain(x2, y2 , z2);
      append (x2, y2, z2) to streamline;
      test the termination condition;
      copy (x2, y2, z2) to x1, y1, z1
   }
end.
```

4. RESULTS

We have used our method to visualize the results (magnetic field and current) of the Ginzburg-Landau equation in a 3D rectangular domain. The Ginzburg-Landau equation is used to compute the critical point of a super conductor[8]. The approximated results of this function are provided by Dr.R.J.Renka[9]. The following figures are of the magnetic fields and currents.

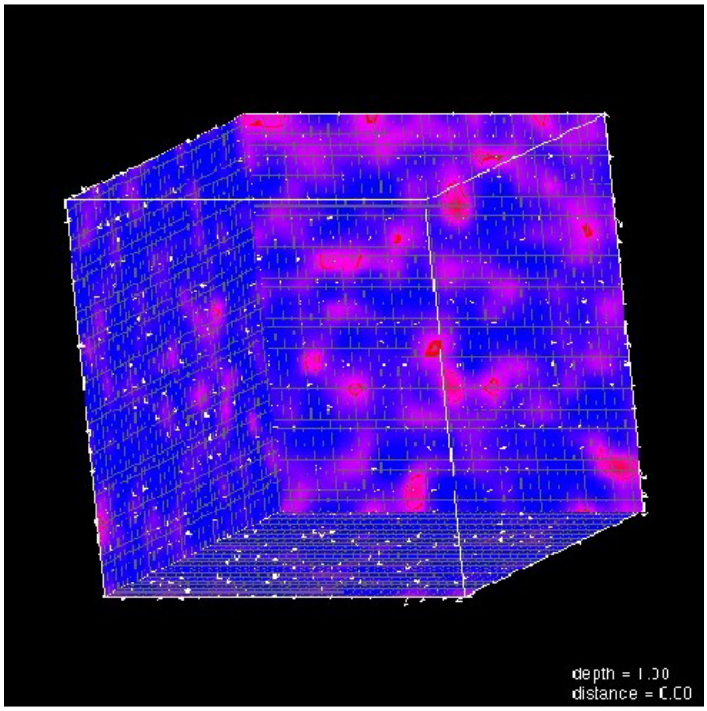


Figure 6. A color-coded image of the current (cross-sections at the x-y, y-z, z-x planes)

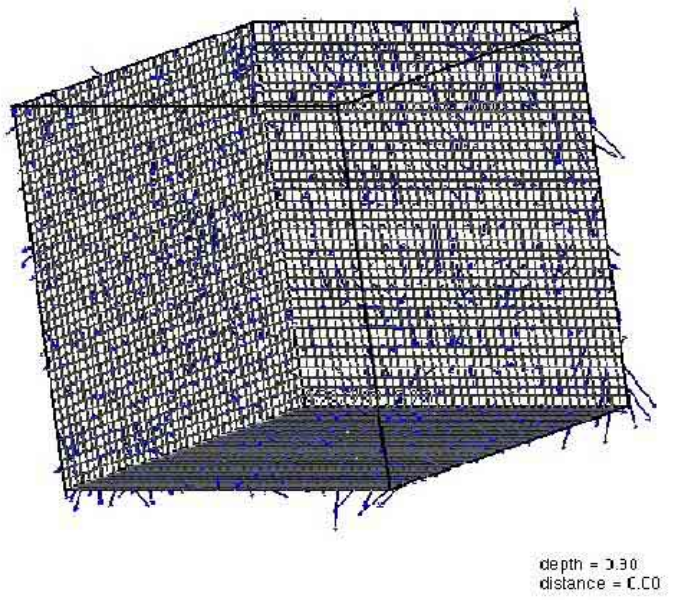


Figure 7. A non-color coded image of the current (cross-sections at the x-y, y-z, z-x planes)

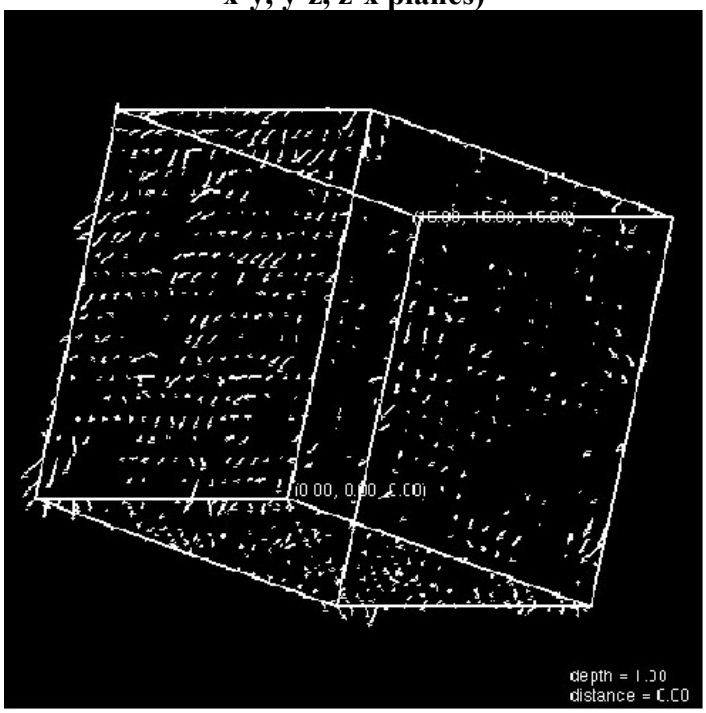


Figure 8. A non-color coded image of the current without grids(cross-sections at the x-y, y-z, z-x planes)

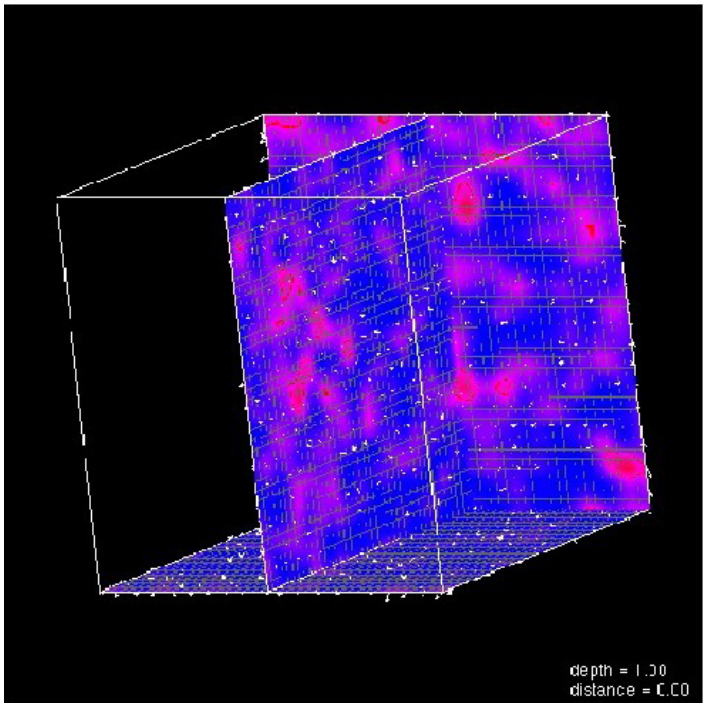


Figure 9. A color-coded image of the current with grids (cross-sections at the y-z, z-x plans, and the cross section in the center of the domain parallel to x-y plane)

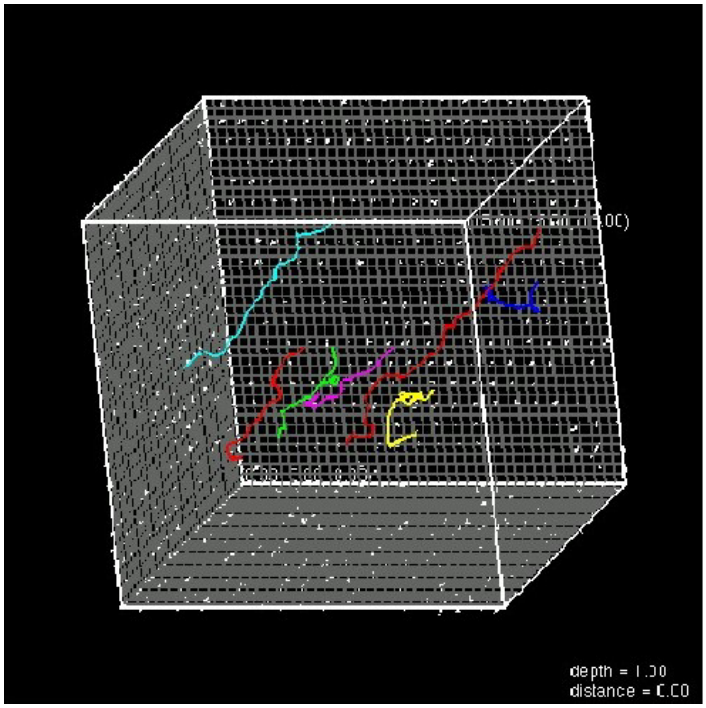


Figure 10. A non-color-coded image of the current with streamlines

From the above figures, we can see that the grids can help the users to establish the concept of a 3D domain. Also, the animation of the cross sections can provide the users with precise information about any grid point in the domain. The streamlines can help users to derive useful information from the data set and give them an overall trend of the vector fields.

5. CONCLUSION

We have presented a simple method to visualize 3D vector fields by displaying the color-coded cross-sections of the 3D vector fields, animating the cross-sections along the axis, and adding arrows and streamlines into the domain. In the implementation of this method, bilinear and the tri-linear interpolation of the grid data have been used.

This method can give us both the local information and part of the global information about the 3D vector field data set. This method, implemented with the interactive user interface, can help users to get a better understanding of the data sets they deal with.

REFERENCES

[1] L. Rosenblum, “Scientific Visualization Advances and Challenges”, Academic Press, 1995, foreword

[2] C. M. Eastman, "Vector versus Raster: A Functional Comparison of Drawing Technologies", *IEEE Computer Graphics & Applications*, 10, 5 (September 1990), pp. 68-80.

[3] Marce Levoy , "Display of Surfaces from Volume Data", *IEEE Computer Graphics and Applications,* Vol. 8, No. 3, May, 1988, pp. 29-37.

[4] Roberto Grosso and Thomas Ertl, "Biorthogonal Wavelet Filters for Frenquency Domain Volume Rendering", *Visualization in Scientific computing ’95*, Springer Computer Science, pp. 81- 95.

[5] M. S. Livingstone, "Arts, Illusion and the Visual System", *Scientific American*, Vol.258, 1988, pp. 78-85.

[6] M. Luckiesh, "Visual Illusions", Dover publication, 1965

[7] Mason. Woo, "OpenGL Programming Guide", Addison Wesley, 1997, pp.594.

[8] Qiang Du, "Analysis and Approximation of the Ginzburg-Landau Model of Superconductivity", *Society for Industry and Applied Mathematics*, Vol.34, No.1, March, 1992, pp.54-81.

[9] J.W.Neuberger and R.J.Renka, "Critical Points of the Ginzburg-Landau Functional on Multiply-connected Domains", *Experimental Mathematics*, Vol.9, No.4, 2000, pp.523-533.