CHAPTER 2. VISUALIZATION TECHNIQUES

In the past, processing speeds and memory limitations bounded the size of electromagnetic problems that could be treated by numerical algorithms. However, as computing power continues to increase so does the capability of computational electromagnetic codes. This allows one to numerically simulate problems that are larger and more complex. The major limitation has quickly become how efficiently one can analyze and interpret the numerical information that is generated. The best method for analyzing large sets of data is through the use of visualization techniques.

Presently, there are a number of commercially available computational codes offering a built-in visualization capability [3]. While these built-in visualization tools are typically easy to use and offer instant feedback of computed results, there are three significant limitations that need to be considered. First, the visualization may restrict the types of field quantities that can be displayed. For instance, the code may allow visualization of input impedance and radiation pattern data while near fields and body currents are not supported. Also, the code may not allow an analysis of the raw data. One may wish to convert from the phasor-domain to the steady-state time-domain (or vice versa) to aid in the interpretation of two-dimensional sets of magnitude and phase data. Finally, most computational packages do not allow the user to improve or expand the visualization tools. For these reasons it is highly desirable to develop a visualization capability that can be fine-tuned to the specific problems that are being analyzed.

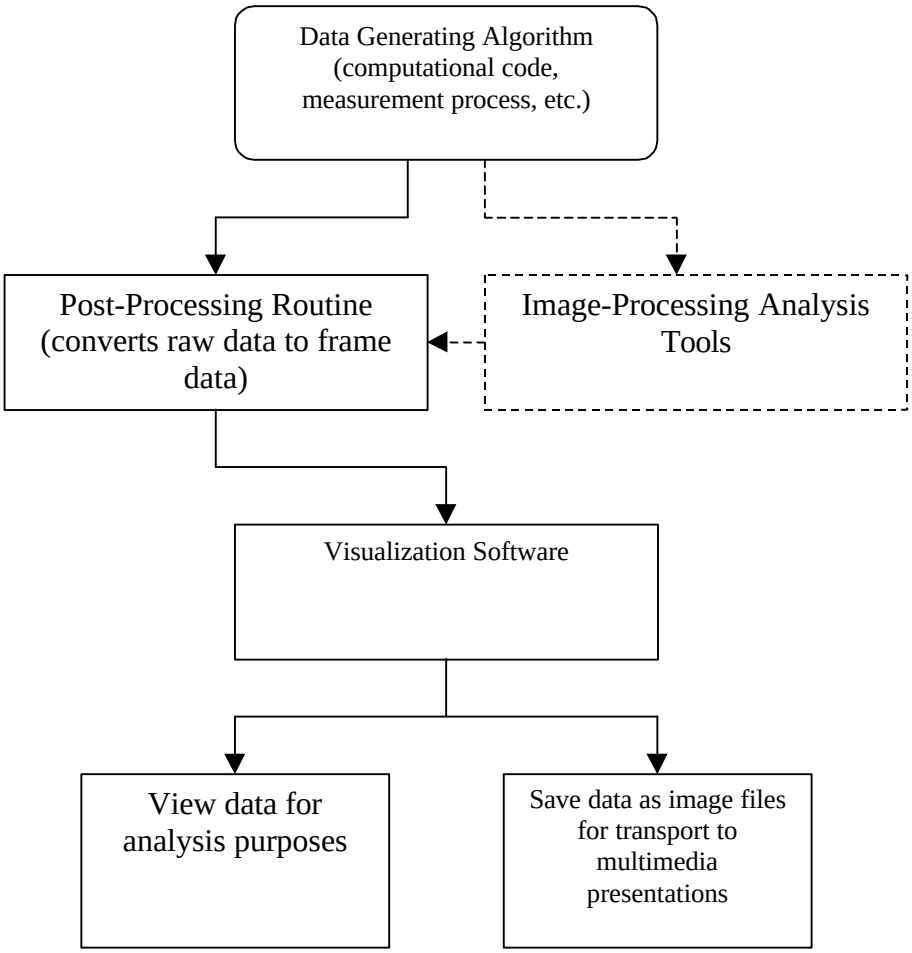

Figure 2-1. Flow chart showing the essential components of visualization capability.

In order to develop a visualization capability, it is first necessary to define the essential components. The flow chart in Figure 2-1 addresses this issue. We start with an algorithm that generates data. In our case this algorithm is a finite-difference time-domain (FDTD) software package. However, the visualization techniques described here are not limited to this application. We could have selected any computational algorithm or empirical process that generates data on an n-dimensional grid. The details of the

data-generating algorithm are beyond the scope of this thesis; however, the interested reader is referred to [4, 5].

The second essential component is a post-processing routine that allows the visualization software to interface with the data-generating algorithm. Typically, the data file generated by a computational code or measurement process cannot be read directly by a visualization package. Therefore, the raw data needs to be converted into a format that the software can read. Details of this component are given in Section 2.1. Additionally, it may be advantageous to discriminate among different modes through the use of image-processing techniques. These methods are detailed in Chapter 4 and are designated in Figure 2-1 by the dashed lines.

The final component is the visualization software. This piece is used not only to analyze the data but also to generate multimedia presentations. Section 2.2 explains some of the concerns that were considered in selecting the visualization package and then gives an overview of the software. The shading techniques employed by the software are discussed in Section 2.3. It is important to understand these techniques because they are typically data smoothing routines, which may distort the data being visualized. Finally, methods for visualizing vector information are explained in Section 2.4.

2.1 Post-Processing Routine

The post-processing routine serves as an interface between the data-generating algorithm and the visualization software. Typically, a visualization package cannot directly read the raw data that is output from a data-generating algorithm. Below is a list

of the three main functions that need to be performed in order to make the raw data file readable.

1. Change data types (ASCII, binary, etc.);
2. Restructure raw data file;
3. Transform data from phasor-domain to time-domain, etc.

The first function that needs to be performed is a conversion of data types. Depending on the data-generating algorithm, the raw data can be in a number of different formats including ASCII, binary, CDF, HDF, and netCDF [6]. In many cases the visualization software will support binary and ASCII data but may not be able to handle some of the less common formats. Therefore, the post-processing routine needs to be able to convert from one data type to another. This function can typically be performed in only a few lines of C or FORTRAN code. Additionally, this function should remove special characters, such as carriage returns, which may "confuse" the visualization software.

The examples presented in this thesis did not require a conversion of data types. The FDTD code output its data in ASCII format, which the visualization software could read directly. We did consider converting from ASCII to binary in order to take advantage of the storage efficiency that binary formatting offers. However, by doing so we would no longer be able to verify the data set by simply viewing it in an ordinary text editor. Since the FDTD code ran on a DOS-based computer and the visualization was performed on a UNIX workstation, it was necessary to transport the data across

platforms. Therefore, we felt that it would be fortuitous to perform a visual check of the data after it was moved from DOS to UNIX.

Now that the visualization software can read the data, the second function needs to ensure that it is read in properly. Figure 2-2 illustrates the differences between columnar data structures and matrix data structures for a two-dimensional data set. Most computational codes and empirical processes provide results in a columnar format while visualization codes commonly expect the input data to be in a matrix structure. Thus, we can see that the raw data file needs to be restructured so it matches the format that the visualization code is expecting. This can typically be accomplished in only a few lines of C or FORTRAN code. Alternatively, some visualization codes use file headers to perform the data parsing. A file header essentially tells the visualization software how to selectively scan a data file so that a single variable is read at a time. The operation of file headers will be explained in terms of the visualization software used to generate the examples in this thesis. While the semantics for a file header are dependent on which visualization code is used, the explanation given here should provide an understanding of the general concepts involved.

In the file header for our visualization software there are three variables that are used for data parsing: skip, offset, and stride. With these three variables it is possible to read data from a wide variety of file structures. The skip variable defines how many lines need to be ignored to arrive at the starting point for the data. This variable allows the visualization software to skip over titling and data labels. Next, the offset variable specifies the number of columns to space over to reach the data starting point. This

Data Coordinate A	Data Coordinate B	Data Values
<Coordinate_A1>	<Coordinate_B1>	<Value_1>
<Coordinate_A2>	<Coordinate_B2>	<Value_2>
<Coordinate_A3>	<Coordinate_B3>	<Value_3>
.	.	.
.	.	.
.	.	.

(a)

Data Coordinate A
<Coordinate_A1> <Coordinate_A2> <Coordinate_A3> **…**
Data Coordinate B
<Coordinate_B1> <Coordinate_B2> <Coordinate_B3> **…**
Data Values
<Value_1> <Value_2> <Value_3> **…**

(b)

Figure 2-2. Two-dimensional data set with (a) columnar structure and (b) matrix structure.

allows for the reading of columnar data. Finally, the stride variable dictates how many pieces of data need to be skipped to arrive at the next data point. For example, we would set skip to 1, offset to 1, and stride to 2 in order to read the data values in Figure 2-2(a).

The final function that needs to be performed is a transformation from the phasor-domain to the steady-state time-domain. This step is necessary since many computational algorithms and empirical processes generate results in a phasor (or time-harmonic) form. For example, the FDTD code that generated the examples in this thesis outputs data in the following phasor format:

$$\vec{E}(x,y) = M_x(x,y)e^{jq_x(x,y)}\hat{a}_x + M_y(x,y)e^{jq_y(x,y)}\hat{a}_y . \qquad (2\text{-}1)$$

The above expression illustrates that each vector component is composed of both magnitude and phase information. In order to analyze and understand the data, these two pieces of information must be combined. Traditionally, the magnitude and phase data were plotted on separate graphs and the scientist or engineer combined the data in his or her mind's eye. This technique is effective for relatively small problems that do not model complex geometries. However, as the size and complexity of the problem increases, the technique quickly becomes cumbersome and confusing. Additionally, most computational routines operate in modulo-2π, which constrains (or wraps) the phase data to the range of $-\pi$ to π. As a result, there are a number of phase jumps (or discontinuities) in the phase information that creates further difficulties when analyzing the data. There are phase unwrapping techniques that remove the phase jumps [7]. However, these techniques introduce errors that can corrupt the data. By converting from the phasor-domain to the steady-state time-domain, one can analyze the data and avoid the two problems discussed above.

Recalling that a time-harmonic field can be written as a complex field vector or as an instantaneous field vector, we can convert the data represented in (2-1) to the steady-state time-domain using the following [8]:

$$\vec{E}(x,y;t) = \mathrm{Re}\left[\vec{E}(x,y)e^{jwt}\right]. \qquad (2\text{-}2)$$

Realizing that one cycle can be looped and animated, if the number of images (N) per cycle are defined, then the following relationship can be used to generate N slices of time-domain data:

$$\vec{E}_n(x,y;t) = \mathrm{Re}\left[\vec{E}(x,y)e^{jn\Delta j}\right]$$
$$where \quad \Delta j = \frac{2p}{N} \qquad (2\text{-}3)$$
$$and \quad n = 0,1,\ldots N-1$$

These slices can be arranged into a three-dimensional array and then animated (a slice at a time) in visualization software. By doing so, the magnitude and phase information is viewed simultaneously. Motion depicts the direction of wave propagation and thus contains the phase information. Being able to watch the fields and currents evolve on an antenna provides us with excellent insight into the radiation process. Chapter 3 will illustrate these ideas using a microstrip patch antenna.

2.2 Overview of Visualization Package

There are a number of commercially available visualization codes for personal computers and workstations. These packages range in functionality as well as in price (from $200 to $10,000+). In order to select one of these packages, it is first necessary to enumerate the requisite operations that the visualization code needs to perform:

- Able to handle large data sets.
- Capable of repeatable processes.
- Allows user to develop custom functions.
- Outputs image formats that are appropriate for use in presentations.

In addition to the above considerations we also tried to select codes which were low in cost and easy to use.

Five packages were evaluated and the results are summarized in Table 2-1. Mathcad was by far the least expensive package; however, it did not fully support three-dimensional (3D) data sets. MATLAB and PV-Wave both could handle 3D data sets; however the graphics capabilities were somewhat lacking. Interactive Data Language (IDL) and Advanced Visual Systems (AVS) were the only two codes that met all of the requirements. We decided to use AVS since the user interface on this package was very straightforward. This was despite the fact that IDL carried a much smaller price tag. Fortunately, the Laboratory for Scientific Visual Analysis at Virginia Tech owns several copies of AVS and allows faculty, staff, and students of the university free access.

Package	Platform	Price	Large Data Sets	Repeatable Processes	Custom Functions	Graphics Capability
Mathcad	Win 95	$350	X	✓	✓	X
MATLAB	Windows	$1,700	✓	✓	✓	X
PV-Wave	Unix	$2,100	✓	✓	✓	X
IDL	Unix	$2,500	✓	✓	✓	✓
AVS	Unix	$12,000	✓	✓	✓	✓

Table 2-1. Summary of visualization software evaluation [9].

AVS is a powerful visualization package that uses a graphical-programming interface. As a result, the user can quickly and easily develop visualization networks that are customized to his or her needs. Figure 2-3 shows the AVS operating environment, which consists of three main areas. The first area is the AVS Module Library that is located at the top right of the screen. The library contains a number of modules, each of which performs a specific operation: data input/output, data conversion, image manipulation, filtering, graphical display, etc. Figure 2-4 shows the module library in

better detail. When developing a visualization network, the user drags modules from the library and drops them in the second area at the bottom of the screen (see Figure 2-3). This area is the workspace, where the visualization network is constructed and modified. Modules are graphically connected to build a visualization network. A simple network is shown as an example in Figure 2-5. The network flows from top to bottom so the read field module will be executed first. This module reads an external ASCII data file. Next, the field to byte module converts the input data to a binary format. A colormap is then defined in the generate colormap module and applied to the data with the colorizer module. Finally, the image viewer module displays the data on the screen for analysis.

Once a network has been developed, it can be saved, reused, or modified. To do so, one uses the AVS Network Editor in the third area at the top left of the screen (see Figure 2-3). The network editor allows the user to read, write, or clear the network. Additionally, it provides tools that allow the user to edit or create custom modules. These tools make it possible to develop unique functions and routines that can be used within a visualization network.

From the brief overview given above, one can see that the AVS software performs all of the requisite operations listed on page 12. In addition, AVS was readily available at the Laboratory for Scientific Visual Analysis at Virginia Tech. Due to the visual-programming environment it is easy to develop visualization networks quickly. In Chapter 3 we will explain specifics of how the AVS software was used to generate the examples presented in this thesis.

Figure 2-3. AVS operating environment.

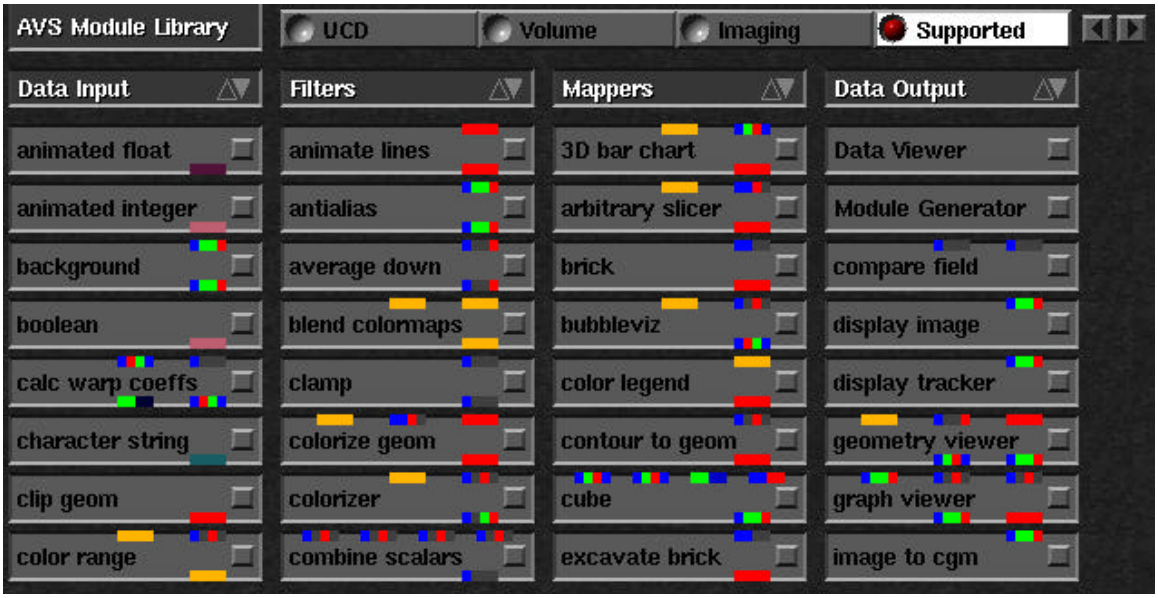

Figure 2-4. AVS Module library.

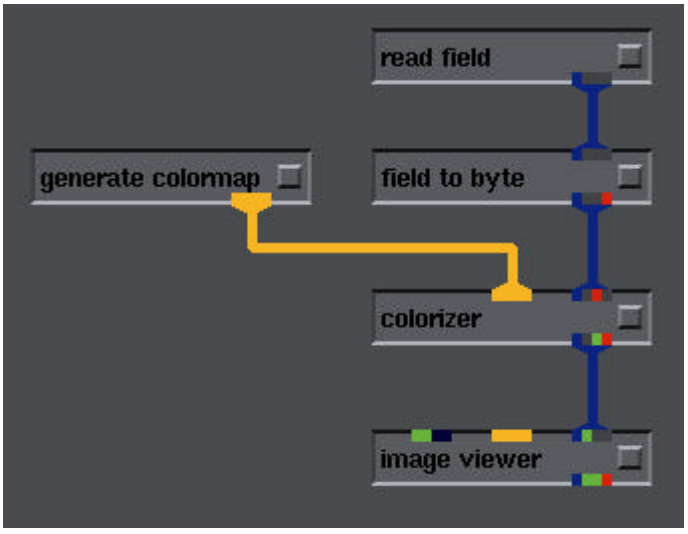

Figure 2-5. Simple visualization network in AVS.

2.3 Shading Techniques used by AVS

Many visualization packages incorporate shading techniques to give visualized objects a more realistic appearance. These techniques are extremely useful when material objects are being visualized (e.g., a three-dimensional rendering of a chair). Typically, the shading techniques can be accessed easily by the user, and it is not uncommon for a visualization code to use a default shading technique at start-up. An important question that needs to be asked is what effect do these shading techniques have on our visualized data. In order to answer this question we first must understand the techniques. AVS supports three different types of shading: flat shading, Gouraud shading, and no lighting. Each of these techniques will be explained below with an emphasis on adverse affects and limitations. For a more complete understanding of shading techniques the reader is referred to [10, 11].

As a bit of a digression, it is first necessary to explain some of the steps involved in converting a two-dimensional data set to a graphical image. In this thesis we are dealing with data that lies on a rectilinear grid as shown in Figure 2-6. In order to convert the data values to an image, the visualization code first assigns a polygon (in our case a rectangle) to each data location. The data values are then displayed by assigning each polygon an offset (surface plot) or a color (raster image). The shading techniques can affect the way in which each data value is mapped to its corresponding polygon, as we shall see below.

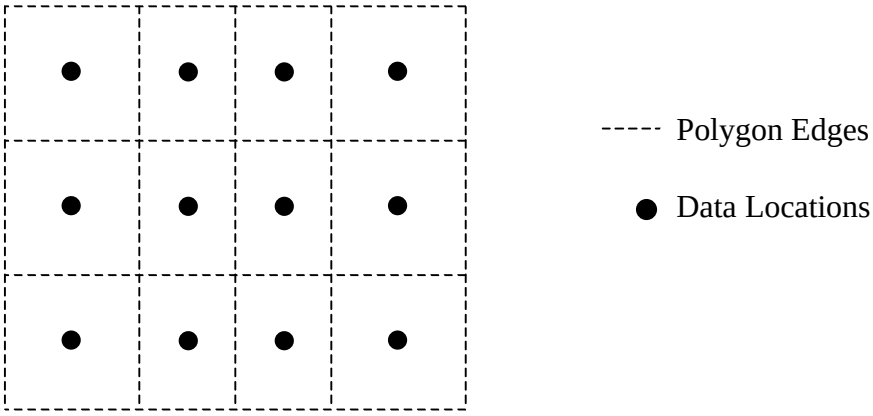

Figure 2-6. Illustration of data locations and polygons.

Flat shading is a fairly simple and computationally efficient technique. It starts with an illumination model that is made up of one or more light sources. Typically, the light source is placed at infinity; although many visualization codes allow the user to custom-define the lighting. Based on this user-defined illumination model, the visualization package determines whether each polygon is in a shaded or lit region. The data value determines the color or offset for each polygon while the shading coefficient determines the intensity. The shading coefficient is then combined with the data value for each polygon. Although this shading technique may sound quite harmless, it can distort our visualized data. Shadowed regions can be responsible for hiding sharp edges or fine details. For this reason this shading technique should be avoided.

The flat shading technique described above sets the entire polygon to the same color and intensity. This tends to produce images with a faceted appearance, which is accentuated by the Mach band effect [12]. This physiological phenomenon accentuates intensity changes at polygon edges where there is a discontinuity in magnitude [10].

Gouraud shading attempts to compensate for this phenomenon by eliminating the intensity discontinuities.

Gouraud's technique uses an illumination model similar to the one above. In this case, intensity values are calculated for each data location. Then, an interpolation is performed along the polygon edges so that there are no discontinuities in intensity. Finally, the interior of the polygon is interpolated along a scan line to fill in the center of each polygon. Figure 2-7 provides an illustrative example. The scan line shown in the figure is normal to the y-axis; this was arbitrarily defined, as the user is typically able to select the orientation of the scan lines in an image.

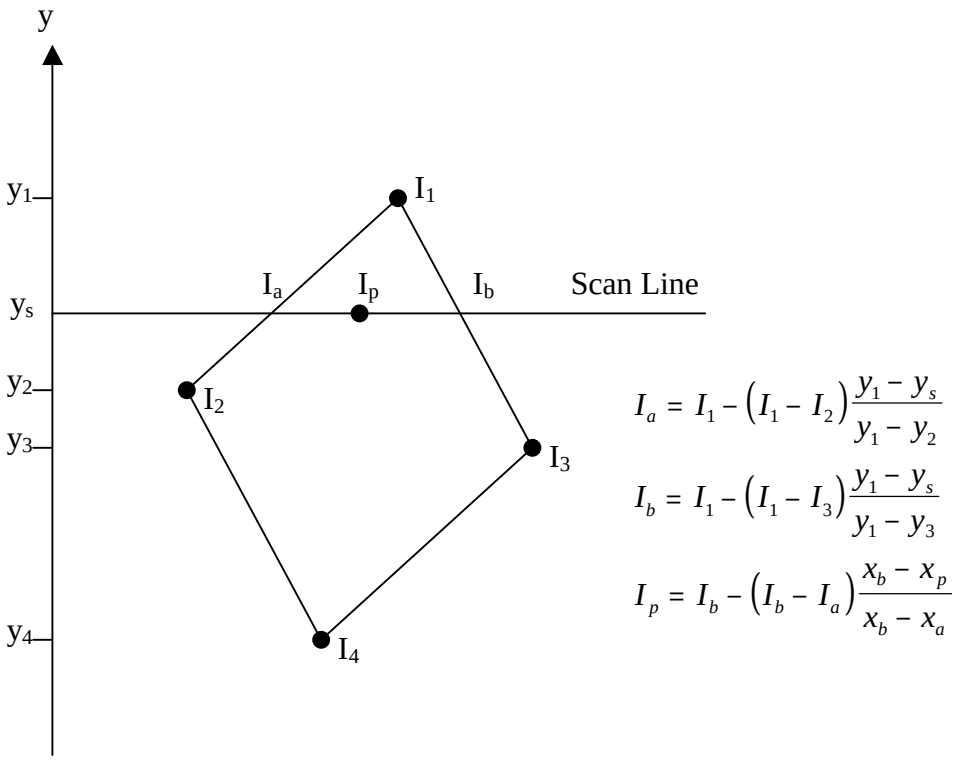

Figure 2-7. Intensity interpolation along polygon edges and scan lines.

As we can see in the above figure, the main difference between flat and Gouraud shading is that the intensity values are interpolated to remove discontinuities at the polygon edges. Consequently, Gouraud shading has the same pitfalls as flat shading. Additionally, the image appearance is now directly dependent on its orientation. This can become a big problem if the data set is rotated in an animation since the image shading may vary as it turns. For these reasons, Gouraud shading should also be avoided.

The final shading technique we will explore is no lighting. This technique is different from the above two methods in that it uses no illumination model (external light sources). Instead, each polygon is treated as a self-luminescent object. Therefore, the appearance of each polygon is solely determined by the corresponding data value. This does result in a faceted appearance for the image; however, this technique does not hide any of the important details. This technique is the best method that we are presented with and will be used in the examples presented in this thesis.

2.4 Visualized Data

The final consideration we need to take into account is how to analyze the data. For the examples presented in this thesis, we must visualize two-dimensional sets of vector information. At each data location we have two data values, the x- and y-components of an electric or magnetic field. Three techniques are considered for viewing the vector data. The first involves viewing the vectors as magnitude and phase information. However, we are now presented with the task of combining the two pieces

of information for a large and complex problem. As was discussed on page 11, this can be cumbersome and confusing.

We next consider a vector plot for the analysis. In this method the data is presented as a vector at each data location. The length of the vector is proportional to the magnitude of the data and the direction of the vector is equal to the phase. This technique is commonly used; however, the data we are visualizing is on a dense grid. Since the data points are so close together, it is difficult to understand the images. We can write a post-processing step to sub-sample (reduce in size) the data set or simply display the data using another method; we have elected the latter of the two.

The final method we have considered displays the data using color raster images. In order to do so we view the vector components separately. We are still presented with the task of combining the information from two images. However, this task is somewhat easier when dealing with vector components rather than magnitude and phase data. The technique converts the component value at each data location to a color. This is done according to a color table, which maps data values to color tones. The definition of a color table is heavily dependent on the researcher and the data being visualized. In the examples presented in this thesis, we have elected to use a linear color table that varied from red to blue. In some cases one may need to use a logarithmic color scale to improve the dynamic range that can be deciphered. Alternatively, one may wish to use a single color and illustrate the data by varying the brightness. Most visualization codes provide a utility for defining a custom color table. Therefore, one can easily tailor a color table to a given application.

Chapter 3 provides an example of the visualization techniques that have been described in this chapter. Animations are presented showing the results of a microstrip patch antenna that was modeled in FDTD. Comparisons are then made between the visualized data and the theory of operation for the antenna to demonstrate that radiation mechanisms can be obtained visually.