1996

Tonitza A Graphics Package for Structural Biology User's Guide

Ioana Maria Martin

Dan C. Marinescu

Report Number:
96-022

http://docs.lib.purdue.edu/cstech/1278

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
TONITZA
A GRAPHICS PACKAGE FOR STRUCTURAL BIOLOGY

USER'S GUIDE

Ioana M. Boier Martin
Dan C. Marinescu

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398

CSD TR-96-022
April 1996
TONITZA
A GRAPHICS PACKAGE FOR STRUCTURAL BIOLOGY
User’s Guide

by
Ioana M. Boier Martin and Dan C. Marinescu
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398

March 1996
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vi</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>vi</td>
</tr>
<tr>
<td>2. THE GRAPHICAL USER INTERFACE.</td>
<td>1</td>
</tr>
<tr>
<td>3. THE INPUT/OUTPUT MODULE</td>
<td>1</td>
</tr>
<tr>
<td>3.1 File Formats</td>
<td>3</td>
</tr>
<tr>
<td>3.1.1 The BRICK Format</td>
<td>3</td>
</tr>
<tr>
<td>3.1.2 The PLANE-BY-PLANE Format</td>
<td>5</td>
</tr>
<tr>
<td>3.1.3 The MAP INTEGER*2 Format</td>
<td>5</td>
</tr>
<tr>
<td>3.1.4 The SGI Format</td>
<td>6</td>
</tr>
<tr>
<td>3.1.5 The TIFF Format</td>
<td>6</td>
</tr>
<tr>
<td>3.2 Reading/Writing Data Files</td>
<td>6</td>
</tr>
<tr>
<td>3.3 Defining Grid Properties</td>
<td>9</td>
</tr>
<tr>
<td>3.4 Saving the Current Session and Exiting Tonitza</td>
<td>10</td>
</tr>
<tr>
<td>4. THE COMPUTATION MODULE</td>
<td>11</td>
</tr>
<tr>
<td>4.1 Data Rotation</td>
<td>11</td>
</tr>
<tr>
<td>4.2 Correlation of Data Sets</td>
<td>11</td>
</tr>
<tr>
<td>4.2.1 Plot Average Density</td>
<td>12</td>
</tr>
<tr>
<td>4.2.2 Compute Optimal Magnification</td>
<td>13</td>
</tr>
<tr>
<td>4.2.3 Refining the Correlation Regions</td>
<td>15</td>
</tr>
<tr>
<td>4.2.4 Compute the Scaling Function</td>
<td>16</td>
</tr>
<tr>
<td>4.2.5 Show/Modify Correlation Parameters</td>
<td>17</td>
</tr>
<tr>
<td>4.3 Composite Map Calculation</td>
<td>18</td>
</tr>
<tr>
<td>4.4 Graphs and Histograms</td>
<td>18</td>
</tr>
<tr>
<td>5. THE VISUALIZATION MODULE</td>
<td>20</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>5.1 2D/3D Representations</td>
<td>20</td>
</tr>
<tr>
<td>5.1.1 Sections</td>
<td>20</td>
</tr>
<tr>
<td>5.1.1.1 Types of Sections</td>
<td>20</td>
</tr>
<tr>
<td>5.1.1.2 The Colormap Editor</td>
<td>25</td>
</tr>
<tr>
<td>5.1.2 Volumes</td>
<td>27</td>
</tr>
<tr>
<td>5.1.2.1 Isosurfaces</td>
<td>27</td>
</tr>
<tr>
<td>5.1.2.2 The Material Editor</td>
<td>29</td>
</tr>
<tr>
<td>5.1.3 Manipulating Sections and Volumes</td>
<td>31</td>
</tr>
<tr>
<td>5.2 Animating Objects</td>
<td>31</td>
</tr>
<tr>
<td>5.2.1 Animating Objects Using the Mouse/Dials</td>
<td>32</td>
</tr>
<tr>
<td>5.2.2 Recording Images and Producing Movies</td>
<td>33</td>
</tr>
<tr>
<td>5.3 Image Processing</td>
<td>33</td>
</tr>
<tr>
<td>6. IMPLEMENTATION NOTES</td>
<td>35</td>
</tr>
<tr>
<td>6.1 The OpenGL Library</td>
<td>35</td>
</tr>
<tr>
<td>6.2 The BX Interface Builder</td>
<td>35</td>
</tr>
<tr>
<td>6.3 The Organization of the Program Modules in Tonitza</td>
<td>36</td>
</tr>
<tr>
<td>6.3.1 The Main Program</td>
<td>36</td>
</tr>
<tr>
<td>6.3.2 Creation Routines</td>
<td>36</td>
</tr>
<tr>
<td>6.3.3 Callbacks</td>
<td>36</td>
</tr>
<tr>
<td>6.3.4 Input/Output Routines</td>
<td>37</td>
</tr>
<tr>
<td>6.3.5 Drawing Routines</td>
<td>38</td>
</tr>
<tr>
<td>6.3.6 Routines for Generating Object Representations</td>
<td>39</td>
</tr>
<tr>
<td>6.3.7 Routines for Object Manipulation/Customization</td>
<td>39</td>
</tr>
<tr>
<td>6.3.8 Computation Routines</td>
<td>41</td>
</tr>
<tr>
<td>6.3.9 Routines for Defining the Fonts</td>
<td>42</td>
</tr>
<tr>
<td>6.3.10 Error Handling Routines</td>
<td>42</td>
</tr>
<tr>
<td>6.3.11 Utility Routines</td>
<td>43</td>
</tr>
</tbody>
</table>

APPENDIX | 44 |

ACKNOWLEDGMENTS | 47 |

BIBLIOGRAPHY | 48 |
LIST OF FIGURES

1. A snapshot of the screen during a Tonitza session. 2
2. Header information window. 7
3. File selection box. 8
4. Data rotation window. 8
5. Dialog window for defining grid properties. 9
6. The correlation steps. 13
7. Average electron density as a function of the particle radius (2D plot). 14
8. Average electron density as a function of the particle radius (3D plot). 14
9. Correlation coefficient as a function of the particle radius. 15
10. A plot of the correlation coefficient as a function of the particle radius. 16
11. The “Show Parameters” window. 17
12. The histogram window. 19
13. Electron density histogram for a Ross River virus structure. 19
14. Dialog window for defining contouring parameters. 21
15. Dialog window for defining continuous scale parameters. 21
16. Dialog window for defining spherical sections. 22
17. A snapshot of the contents of the command area for sections. 23
18. Dialog window for defining contour stacks. 24
19. Stack of mask contour maps for a Coxackievirus B3 asymmetric unit. . . . . 24
20. Spherical sections viewed along the two-, three-, and five-fold axes. . . . . 25
21. The colormap editor. .................................................. 26
22. Dialog window for defining the isosurface parameters. ....................... 28
23. A snapshot of the command area for volumes. ................................ 29
24. The material editor. .................................................... 30
25. Dialog window for printing images. .................................... 32
Tonitza is a graphics package for Structural Biology which provides capabilities for visualizing scientific data sets in various two- and three-dimensional representations and for combining visualization with specific computations to allow the biologist to interpret and debug his/her data. This guide contains examples and step-by-step instructions for using the package.
1. INTRODUCTION

Tonitza is an interactive graphics package which is relatively easy-to-use and easily extensible to include new features. It has been specifically designed to deal with large data sets, such as those produced in X-ray crystallography and electron microscopy experiments. Although it is aimed at the field of Structural Biology, Tonitza can be used to visualize any vector field data.

Tonitza consists of a graphical user interface (GUI), input/output, visualization, and computation modules. This document provides step-by-step instructions for using the package. All examples are based on data sets representing Ross River or Coxackie B3 virus structures [CHEN95], [MUCK95].

2. THE GRAPHICAL USER INTERFACE

Tonitza incorporates a variety of interface styles. The main style is that of a direct-manipulation user interface [FOLE90], in which the objects and attributes that can be operated on are represented visually and operations are invoked by actions performed on the visual representations, typically using the mouse. However, this interaction style is not sufficient by itself and other interface styles such as menus are also included. Figure 1 shows a snapshot of the screen during a Tonitza session.

3. THE INPUT/OUTPUT MODULE

The Input/Output (I/O) module is responsible for reading/writing data files from/to the disc, for automatic file format recognition, for conversion between formats, and for handling I/O errors.
Figure 1. A snapshot of the screen during a Tonitza session.
3.1. File Formats

Conceptually, each file consists of a header followed by data. The header usually contains a magic number (for format identification), some comment about the contents of the file, the dimensions of the file and other statistical information (e.g. minimum and maximum data values, background value, etc.). You can display the header information by choosing the Show Header Info option from the File or Options menus. Figure 3 shows the contents of the header window for a file in the MAP INTEGER*2 format. The dialog box at the top of the window allows you to specify the name of a file for which you wish to display the header information. You can type the full path of the file in the dialog box or you may press the “...” button and select a file name from a list of files already opened. The data follows the header and it may be compressed or not (if it is compressed, some compression information may be stored between the header and the actual data). The data values correspond to a two- or three-dimensional vector field. The rest of the section describes in detail the formats accepted in Tonitza version 2.0. Note that the program can be easily modified to include other formats.

3.1.1. The BRICK Format

A file in the BRICK format [ROSS94] consists of a header (64K) and a set of sequentially stored bricks of data. A brick is a 3D volume consisting of a number of consecutive grid points. For each grid point, the information is packed into two bytes: 10 bits electron density and 6 bits mask. Note that for h-cell data in this format only electron density information is be available.

Given a 3-D mesh of \( na \times nb \times nc \) grid points, from \( iix \) to \( ifx \) in the x-direction, from \( iiy \) to \( ify \) in the y-direction and from \( iiz \) to \( ifz \) in the z-direction, we have: \( na = ifx - iix + 1, nb = ify - iiy + 1, \) and \( nc = ifz - iiz + 1. \) A brick is a 3-D volume consisting of \( bx \times by \times bz \) consecutive grid points. Currently \( bx = by = bz = 16. \) There are 4096 grid points within a brick and the storage space occupied by a brick is 8K bytes. The grid points within a brick are stored in the \( x \rightarrow y \rightarrow z \) order.

We can look at the bricks as a 3-D structure consisting of layers of bricks stored in
the $x \to y \to z$ order. Call "$ibrick"", "$jbrick"", "$kbrick" the number of bricks in the $x$, $y$ and $z$ direction respectively. The values "$ibrick"", "$jbrick"", "$kbrick" are computed as follows:

\[ ibrick = (na - 1)/bx + 1, \quad jbrick = (nb - 1)/by + 1, \quad kbrick = (nc - 1)/bz + 1, \]

$ijbrick = ibrick \times jbrick$. The brick reference point is the brick point closest to the origin of the 3-D brick space. Given the brick with 3-D coordinates $(ibid, jbid, kbid)$ its reference point has the 3-D coordinates $(istart, jstart, kstart)$ given by:

\[ istart = bx \times (ibid - 1) + iix, \]
\[ jstart = by \times (jbid - 1) + iiy, \]
\[ kstart = bz \times (kbid - 1) + iiz. \]

The relation between local and global coordinates of a grid point is as follows. Assume that you are given $(i, j, k)$ the local coordinates of a point in brick "$bid" with the reference point $(istart, jstart, kstart)$. Then its global coordinates $(iglobal, jglobal, kglobal)$ are given by:

\[ iglobal = i + istart - 1, \quad jglobal = j + jstart - 1, \quad kglobal = k + kstart - 1. \]

The bricks are stored sequentially in the $x \to y \to z$ order.

<table>
<thead>
<tr>
<th>Brick 1</th>
<th>Brick 2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The brick with 3-D brick coordinates "$ibid", "$jbid", "$kbid" is stored sequentially as brick "$bid" with:

\[ bid = (k - 1) \times ibrick \times kbrick + (j - 1) \times ibrick + i. \]

The first layer of bricks ($z=1$) contains $ibrick \times jbrick$ bricks:

1, 2, .... $ibrick$,

$(ibrick + 1, .... 2 \times ibrick),$

$(ibrick \times (jbrick - 1) + 1 ....... ibrick \times jbrick)$

The total number of bricks is: $nbr = ibrick \times jbrick \times kbrick$.

Note: The brick format is used to store both the p-cell (which contains either the mask or the electron density and the mask) and the h-cell which contains only the electron density.
For the h-cell the computation of the mesh geometry is similar to the one described above for the p-cell: 

\[ mx = i_fh - i_ih + 1, \quad my = i_fh - i_ih + 1, \quad mz = i_fh - i_ih + 1, \quad \text{ibrickh} = (mx-1) / bxh + 1, \quad \text{jbrickh} = (my - 1) / byh + 1, \quad \text{kbrickh} = (mz - 1) / bzh + 1, \quad \text{ijbrickh} = \text{ibrickh} \times \text{jbrickh}, \quad \text{nbrh} = \text{ibrickh} \times \text{jbrickh} \times \text{kbrickh}. \]

3.1.2. The PLANE-BY-PLANE Format

A file in this format [ROSS94] consists of a header (64 KB) and a set of sequentially stored y-planes of data. This data format is used to export results produced by envelope to fftinv or to import from fftsynth new electron density data.

3.1.3. The MAP INTEGER*2 Format

In this case, the data file consists of a header and a set of sequentially stored "sections" of data [BAKE95]. Each data record contains a row of density values. This structure is summarized in the table below:

<table>
<thead>
<tr>
<th>REC #</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I*4</td>
</tr>
<tr>
<td>2</td>
<td>18A4</td>
</tr>
<tr>
<td>3</td>
<td>3I*4, 5F</td>
</tr>
<tr>
<td>4</td>
<td>6F</td>
</tr>
<tr>
<td>5</td>
<td>3F, 6I*4</td>
</tr>
<tr>
<td>6</td>
<td>ncol I*2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>nrow*ncol+6</td>
<td>...</td>
</tr>
</tbody>
</table>
3.1.4. The SGI Format

This is a generic bitmap format used for storing black-and-white, gray-scale, and color images [IMAG94]. SGI files may contain compressed or uncompressed data. In the former case, the file consists of a 512 bytes header, a scanline offset table and compressed image data. In the latter case, the image data appears immediately after the 512 bytes header. The format of the header is described in what follows:

- **SHORT Magic** -- identification number (474)
- **CHAR Storage** -- compression flag
- **CHAR Bpe** -- bytes per pixel
- **WORD Dimension** -- number of image dimensions
- **WORD XSize** -- width of image in pixels
- **WORD YSize** -- height of image in pixels
- **WORD ZSize** -- number of bit planes
- **LONG PixMin** -- smallest pixel value
- **LONG PixMax** -- largest pixel value
- **CHAR Dummy1[4]** -- not used
- **CHAR ImageName[80]** -- name of image
- **LONG ColorMap** -- format of pixel data
- **CHAR Dummy2[404]** -- not used

3.1.5. The TIFF Format

This is a versatile bitmap format that supports numerous data compression schemes and it is used for data storage and interchange [IMAG94]. TIFF data files are organized into three sections: the Image File Header (IMH), the Image File Directory (IFD), and the bitmap data. A TIFF file may contain multiple images.

3.2. Reading/Writing Data Files

To read a data file you must choose the **Open Input File** option from the **File** menu. A file selection box allows you to browse the files and directories in your file system and
to select a data file for opening. Figure 3 shows a snapshot of the file selection box. Version 2.0 allows up to three input files to be opened simultaneously. You may also close a file by choosing the **Close Input File** option from the **File** menu.

Data generated in a Tonitza session may be saved into a file using the same file selection mechanism to specify the name of the file. In version 2.0 this feature is available for saving difference map files. To save such a map, you must choose the **Correlation Coefficient** option from the **File** menu. The **Compute** menu in the new menu bar contains the **Write to File** option needed for this operation.
Figure 3. File selection box.

Figure 4. Data rotation window.
3.3. Defining Grid Properties

Each input file format has associated a default number of properties, that is, values corresponding to each grid point. For example, files in MAP INTEGER*2 format have only one default property: the electron density, files in BRICK format have two default properties: electron density and particle masks. Tonitza allows you to replicate the existing properties, to redefine their ranges and default names. This feature is useful for visualization purposes, if you wish to visualize the same property under different representations that cannot be displayed simultaneously otherwise. The Define Grid Properties option from the File menu allows you to specify the number of properties you wish to associate with a particular input file, the name and range of values for each such property. Figure 5 shows the dialog window for defining the grid properties.

![Figure 5. Dialog window for defining grid properties.](image-url)
3.4. Saving the Current Session and Exiting Tonitza

A Tonitza session may be terminated by pressing the Exit button from the File menu. A small window allows you to confirm or cancel the termination and also to save some of the current settings to be used as defaults in the next session. The current settings are saved in the file _T.defaults and they are as follows: the path where the last input data file that was opened is located, the coefficients of the linear transformation to be used for composite maps, the magnification parameters (range and step), and the angles for data rotation.
4. THE COMPUTATION MODULE

The computation module allows you to carry out various computations, such as: data rotation, correlation of two data sets, scaling, composite map calculation, graphs, and histograms.

4.1. Data Rotation

Given a rotation axis and an angle of rotation, Tonitza allows you to obtain a new data set in which the values are obtained by trilinear interpolation from the original data set and reflect the new orientation of the particle with respect to the grid. This feature is useful if you want to extract subvolumes based on symmetry elements, to visualize the data volume in different orientations, or to slice the data volume with arbitrary planes. You may perform data rotation by choosing the **Rotate Map** option from the **File** menu. Figure 4 shows the data rotation window. The two dialog boxes at the top of the window allow you to specify the name of the file and the property for which the rotation is to be applied. These may be specified either by typing the names in the corresponding text dialog boxes or by pressing the "..." button and selecting the desired element from a list of existing file names or properties, respectively. The rotation may be defined by specifying a combination of rotations about the X, Y, and Z axes (in this order, if two or more are toggled). A rotation about an axis is performed only if the corresponding toggle button is on. A rotation angle may be specified by typing a value (in degrees) in the text dialog box next to the toggle button for each of the X, Y, and Z axes.

4.2. Correlation of Data Sets

The correlation of two data sets may also be computed. This feature has been designed according to the procedure used in electron microscopy for fitting structures separately obtained [BAKE95]. That is, given two maps A and B, the correlation procedure aims to determine an optimal magnification factor used to reinterpolate B so that the trans-
formed B data set best fits A. A measure of the best fit is given by the correlation coefficient of the two data sets. The correlation coefficient may be computed for the entire volume or just for certain slabs within it. When you choose the Correlation Coefficient option from the File menu, the contents of the main window and the main menu change to accommodate the steps necessary for computing the correlation of two data sets. Note that this option always requires at least two input files to be open. The names of the two files and the properties to be correlated have to be specified in the corresponding text dialog boxes. By default, these are the first two files that are open and the first property in each of the files. Figure 6 shows the steps involved in the computation of the maximum correlation coefficient (and hence, of the optimal magnification factor). A description of each of these steps follows.

4.2.1. Plot Average Density

This step is useful when you would like to correlate only the density values within certain concentric spherical shells, rather than in the entire data volume. A plot of the average electron density versus the radius of the particle allows you to define the correlation regions. By default, correlation will be performed in the entire volume. Figure 7 shows a 2D plot of the average electron density as a function of the radius in the case of a native Ross River virus structure [CHEN95]. The green rectangles define the correlation regions (in this case, between radii 23.7 and 35.0 and between 37.1 and 48.4). You may define up to 10 such regions by clicking with the left mouse button in the yellow area. The Clear button below the plot allows you to remove all existent regions and restart defining them. The 3D button allows you to obtain a 3D representation of the average density as a function of the radius. Figure 8 shows such a representation, in which the average density values are mapped onto a grayscale. The height of the surface at every radius is proportional to the average density value at that radius. The correlation regions are displayed in green. To return to the 2D representation you have to press the 2D button. The radii chosen to define the correlation regions are displayed at the left of the plot.
4.2.2. Compute Optimal Magnification

Once you have defined the correlation regions, you can compute an optimal magnification coefficient to be used for fitting the two data sets you are working with. Tonitza allows you to define a range and a step for the magnification coefficient and to compute the correlation coefficient using each of the magnification values. The one which yields the largest correlation coefficient is retained as the optimal magnification coefficient. In addition to the correlation regions defined in the previous step, you may also restrict your
Figure 7. Average electron density as a function of the particle radius (2D plot)

Figure 8. Average electron density as a function of the particle radius (3D plot)

correlation domain to X, Y, Z sections and/or slabs by using the slider bars provided in the
optimal magnification window. The computation starts when you press the Start button and a 2D plot of the correlation coefficient as a function of the magnification factor is displayed. The maximum value of the correlation coefficient and the corresponding magnification factor are printed beneath the plot as shown in Figure 9.

Figure 9. Correlation coefficient as a function of the particle radius.

4.2.3. Refining the Correlation Regions

After you have determined an optimal magnification factor, Tonitza allows you to refine the correlation regions by redefining them using a plot of the correlation coefficient as a function of the particle radius. The optimal magnification factor computed in the previous step is now used for the correlation. The regions may be redefined in a manner similar to the one described in section 4.2.1. by clicking the left mouse button on the current plot. Figure 10 shows such a plot obtained by correlating two Ross River virus structures.
4.2.4. Compute the Scaling Function

The first three steps of the correlation procedure affect the scaling of one grid with respect to the other. In addition, you may also want to fit the electron density ranges in the two data sets. Tonitza allows you to compute a linear transformation to be applied to the density values in one data set so that the average density in the two data sets being fitted is the same. Two coefficients \( c_1 \) and \( c_2 \) are computed based on the average values \( \overline{A} \) and \( \overline{B} \) of the two data sets such that: \( \overline{A} = c_1 \overline{B} + c_2 \). The values of \( c_1 \) and \( c_2 \) are given by the following expressions: 

\[
  c_1 = \frac{\sum (b_i - \overline{B}) \times (a_i - \overline{A})}{\sum (b_i - \overline{B})^2}, \quad c_2 = \overline{A} - c_1 \overline{B}.
\]

To start the computation of the scaling parameters \( c_1 \) and \( c_2 \) you have to press the Start Computation button. The steps of the computation will be highlighted in red as the computation progresses. Note that the scaling parameters determined here will also be used as default values for the composite map calculation.

![Figure 10. A plot of the correlation coefficient as a function of the particle radius.](image)
4.2.5. Show/Modify Correlation Parameters

At every step in the correlation procedure, you can obtain the values of all the parameters involved by pressing the **Show Parameters** button. Figure 11 shows the contents of the parameters window. Some of the parameters in this window are editable and you can change their values by hand.

![Figure 11. The “Show Parameters” window.](image-url)
4.3. Composite Map Calculation

This feature is available from the File menu (the **Composite Map** option) and it allows you to compute any linear combination of two given maps. The names of the two data files and the corresponding properties, as well as the coefficients \( c_1 \) and \( c_2 \) of the linear transformation may be specified by typing their values in the appropriate dialog boxes. The actual transformation is: \( \text{New Map} = \text{Map}_1 - (c_1 * \text{Map}_2 + c_2) \). If a scaling operation has been previously performed, then the default values of \( c_1 \) and \( c_2 \) are those determined by the scaling procedure. Otherwise, the default values are \( c_1 = 1, c_2 = 0 \). The new map is stored internally under the name **difference** and it may be visualized as any other input data file. **Note** that it counts as an input data file, so you may want to close it when you don’t need it any longer. The **Write to File** option from the Compute menu allows you to write this map to the disc. The output file format depends on the input format of the two files from which it was computed. For example, if the input format was MAP INTEGRER*2, then the output format is MAPI2T (T stands for Tonitza) and it has the same structure, with the exception of the magic number and of the fact that the data is not compressed.

4.4. Graphs and Histograms

Graphs are used in Tonitza to support other features such as the steps of the correlation procedure. Figures 7, 8, 9, and 10 are examples of such graphs.

Histograms are a useful tool which provides you with statistical information about your data. The **Histogram** option can be found in the Options menu for both sections and volumes. Histograms may be computed for two types of properties: **discrete** (such as masks) and **continuous** (such as electron density). The user may choose between these two options by setting the appropriate toggle button in the Histogram window (see Figure 12). In the case of continuous histograms, the number of equally spaced intervals to be used must also be specified. Figure 13 shows a histogram for a Ross River virus structure [CHEN95].
Figure 12. The histogram window.

Figure 13. Electron density histogram for a Ross River virus structure.
5. THE VISUALIZATION MODULE

The visualization module allows you to display your data in various two- and three-dimensional representations, to animate and customize the objects displayed.

5.1. 2D/3D Representations

Two types of representations are available in Tonitza: sections and isosurfaces. They are described in what follows.

5.1.1. Sections

The Sections option from the Display menu allows you to visualize sections of your data. This type of representation provides support for slicing the data volume with planes or spheres and for visualizing the data in these slices.

5.1.1.1. Types of Sections

Planar Sections (the first option in the Sections menu) are sections parallel to the planes of coordinates through the data volume. The data in such sections may be visualized as contours at various levels or as a continuous scale map.

Upon selecting the Contours option from the Planar menu, a new window, requesting information about the contour levels appears on the screen. Figure 14 shows the format of this window. The information you must fill in refers to the name of the data file, the property to be contoured, the number of contour levels, the value and the color of each level. Maximum 16 contour levels are allowed in version 2.0 of Tonitza. To confirm the values chosen for the number of levels and for each level value, you must press the OK button next to the corresponding dialog box. To choose a color for a particular level, you may type a valid color name in the dialog box provided and then confirm your choice by pressing the corresponding OK button, or you may choose a color from a list of colors which pops up when you press the "..." button next to the color dialog box. To apply a
color selected from this list you must press the **Apply** button. The name of the color will also appear in the color dialog box. The **Next** button allows you to define the parameters for the next contour level. When the last contour level is completely defined, the **Next** button changes its label to **Done** and by pressing it you dismiss the window and start the computation of the contours.

**Continuous Scale** maps are a different kind of representation, in which data values are mapped to colors. This option is also available from the **Planar** menu. Figure 15 shows the dialog window corresponding to this type of map. The only information needed in this case is the name of the data file and the property which is to be mapped. The mapping takes place when you press the **Accept** button in this window.

![Figure 14. Dialog window for defining contouring parameters.](image1)

![Figure 15. Dialog window for defining continuous scale parameters.](image2)
Initially, the data values are mapped linearly on a gray scale form black (the smallest value) to white (the largest). This mapping may be altered using the colormap editor. For details of how to use the colormap editor, see section 5.1.1.2 of this document.

**Spherical sections** (the second option in the **Section** menu) may be displayed for cubic data volumes (that is volumes for which the number of grid points in each of the X, Y, and Z directions is the same and for which all angles are $90^\circ$). The center of a spherical section is the center of the cubic volume. Figure 16 shows the dialog window for defining spherical sections. Besides the file name and the property for which the sections are computed, you must also specify an interval for the radii of the spheres. This interval is defined by the first and the last radii for which spherical sections are to be displayed and by a positive increment (step). All spheres are approximated with polyhedra and the **Sampling** parameter defines the number of vertices of such a polyhedron along one large hemicircle of the sphere. If the sampling is too coarse, the polyhedral nature of the sphere will be visible. If it is too fine, manipulating the spherical sections in real time will become difficult. The recommended sampling is one for which the number of vertices is equal to the number of grid points along one of the volume edges. The sections are computed when you press the **Accept** button.

![Figure 16. Dialog window for defining spherical sections.](image-url)
The only representation available for spherical sections is the continuous scale map. The data values on the sphere are interpolated from the data values at grid points and they are subsequently mapped to colors, in the same way as for planar continuous scale maps. The mapping may also be changed using the colormap editor.

Once the sections (planar and/or spherical) have been computed they can be displayed in the main drawing area, swept through the entire range of sections available, translated, and rotated. X, Y, and Z planar sections may be displayed simultaneously or individually, by selecting the corresponding toggle button in the command area to the right of the drawing area (see Figure 17).

Figure 17. A snapshot of the contents of the command area for sections.

The R toggle button allows you to turn on/off the spherical sections. A set of slider bars is provided for displaying sections at various positions within the volume. The planar sections may be displayed at consecutive grid positions along the X, Y, and Z directions. The spherical sections may be displayed from the first to the last section specified, in increments equal to the step. The Auto toggle button enables the automatic mode for spherical sections in which the R slider bar and the display are updated automatically, without any action from the user. When the Axes toggle button is on, the axes of coordinates are displayed in the drawing area, to help you orienting the objects displayed. The Stack toggle button refers to contour maps and it allows you to display several such maps superimposed. The parameters of the stack may be defined interactively, as shown in fig-
The orientation of the stack may be specified using one of the $X$, $Y$, or $Z$ toggle buttons. The sections in the stack are taken from the First to the Last in increments of Step. Figure 19 shows a stack of masks for a Coxackievirus B3 asymmetric unit.
Spherical sections may also be replicated using the **Replicate** button. This feature is especially useful for data sets representing particles with icosahedral symmetry. The replicas allow you to visualize the particle in the standard orientation (along a two-fold axis), along the three- and five-fold axes simultaneously. Figure 20 shows three such spherical sections in the case of a Ross-River virus particle.

![Spherical sections viewed along the two-, three-, and five-fold axes.](Ross-River virus)

**Note:** if several files are open, the option menu at the bottom of the command area allows you to choose between them (options First File, Second File, etc.) The slider bars are always related to the currently selected file.

5.1.1.2. The Colormap Editor

The colormap editor is available from the **Options** menu of the **Sections** window,
also two options: **Undo**, which restores the color mapping to the state before the last color cell selection, and **Place Knots** which restores the initial state (that of a linear gray scale).

The **graph area** consists of two parts: the lower half contains the colors currently stored in the colormap in the form of vertical rectangles (the overall appearance is that of a smoothly varying color scale); the upper half contains a graph of the red, green, and blue color components, for each of the colors in the colormap. Initially, these values vary linearly between 0 and 1 for each of the R, G, B components. The **color cell area** contains a square cell for each color in the graph area. The total number of colors depends on the number of colors that can be allocated on a particular machine. At first, Tonitza tries to allocate 256 colors for the colormap editor. If it cannot, then it tries to allocate 128, 64, etc. The minimum number of colors required is 4. Each color cell is selectable by clicking on it with the left mouse button. The currently selected color cell is marked by a red "X" and its number appears under the color cell area. Every time you select a color cell, a "knot" is defined, and it appears in the graph area as a black vertical line above the corresponding color. The **slider bars** (one for each of the R, G, and B components) allow you to alter the color corresponding to the current knot. This change is reflected in both the graph and color cell areas. The R, G, B components are no longer linearly varying between the first and the last colors, but they are piecewise linearly varying between consecutive knots.

5.1.2. Volumes

The **Volumes** option from the **Display** menu allows you to visualize volumes or sub-volumes of your data.

5.1.2.1. Isosurfaces

The only volumetric representation available in section 2.0 is the **isosurface**. An isosurface is the set of all points in a volume with a given scalar value [FOLE90]. When you select the **Isosurface** option from the **Volume** menu, a dialog window like the one shown in Figure 22 is displayed. It allows you to specify the name of the input data file and the property for which the isosurfaces are to be computed, the (sub)volume you are interested
under the name Colormap. Figure 21 shows the structure of the editor. It consists of a menu, a graph area, a color cell area, and a set of slider bars. The menu has several sub-menus, some of which are not implemented in version 2.0. The active sub-menus are Map and Edit. The Map menu has two options: Save, which allows you to save the current colormap in a file, and Close, which pops down the editor window. The Edit submenu has

Figure 21. The colormap editor.
in, the number of isolevels, and a value for each level. The (sub)volume of interest can be specified with the help of six slider bars: \([X_{\text{min}}, X_{\text{max}}] \times [Y_{\text{min}}, Y_{\text{max}}] \times [Z_{\text{min}}, Z_{\text{max}}]\). The Step dialog box allows you to specify an increment (by default 1) to be used between the min and max values along each of the X, Y, and Z directions (measured in grid points). For example, if the increment is equal to 2 then only every second grid point along each direction will be considered when computing the isosurface (this is equivalent to coarsening the grid by a factor of two in each direction).

Figure 22. Dialog window for defining the isosurface parameters.
The (sub)volume may be further restricted to concentric shells around the origin by specifying an inner and an outer radius in the two shell text areas. The computation of the isosurfaces starts as soon as you press the Accept button. Initially the isosurfaces are displayed as wireframe. In this representation the surfaces are displayed as if they were made of wire, with only the edges of the component polygons showing. The other representation available is Gouraud shaded, in which the interiors of the polygons are filled with colors influenced by the light sources in the scene. You can switch between these two representations and alter the properties of the materials the surfaces appear to be made of by using the Material Editor (see section 5.1.2.2) available from the Options menu.

The command area in the case of the volumes is shown in Figure 23. A small drawing area at the top shows the position of the selected (sub)volume with respect to the entire volume and the axes of coordinates reveal the current orientation of the (sub)volume displayed.

![Figure 23. A snapshot of the command area for volumes.](image)

If the Box toggle button is on, a red wire box is displayed around the selected subvolume in the main drawing area.

5.1.2.2. The Material Editor

The Material Editor is available from the Options menu of the Volumes window. Figure 24 shows the structure of the editor. It consists of a display area, a dialog area, and
a set of slider bars. The *display area* shows an ellipsoid made of the current *material*. Its role is to give you an example of how the material you have defined looks like before you apply it to the isosurfaces. The dialog area consists of two parts: the upper part allows you

![Figure 24. The material editor.](image)
to specify the name of the input file, the property to which the material is to be applied, and to choose between a wireframe or a smoothly shaded representation; the lower part contains a toggle button for each of the isosurface levels that have been defined for the current property. The slider bar area changes its contents depending on this representation. For a wireframe object, the only parameters to be defined are its color and its opacity. In this case, the slider bar area contains only the Red, Green, Blue, and Alpha slider bars. If a shaded representation is to be specified, in addition to the four slider bars previously mentioned, a Shininess slider bar is displayed together with a set of three toggle buttons: one for defining the Ambient, one for the Diffuse, and one for the Specular light reflected by the material. The slider bar area also contains a set of four buttons: Reset, for resetting all the material parameters to their default values, Apply, for applying the current material to all isosurface levels that are toggled, Close, for dismissing the material editor window, and Help, not implemented in version 2.0.

5.1.3. Manipulating Sections and Volumes

After an object is displayed, you can make it larger or smaller (using the Zoom and Shrink buttons from the command area), you can animate it (see section 5.2), or you can restore its initial position (the Reset button from the command area). The Clear button from the Sections/Volumes menu destroys all objects created and clears the main drawing area. The current contents of the drawing area may be recorded (using the Record button in the command area) or printed (the Print option from the Options menu). Section 5.2.2 provides a detailed description of how to record and process images in Tonitza. To print the contents of the drawing area you must specify the name of the printer and whether you would like a color or grayscale printout (only color available in version 2.0). Figure 25 shows the format of the printing dialog window.

5.2. Animating Objects

The objects produced in Tonitza may be animated in several ways: using the slider bars (as described in section 5.1.1.1), using the mouse or the dials, and by concatenating a
number of frames into a movie file and playing it.

![Figure 25. Dialog window for printing images.](image)

5.2.1. Animating Objects Using the Mouse/Dials

An object displayed in the drawing area can be rotated using the middle mouse button. To rotate the object you must keep the button pressed and to drag the mouse with movements similar to those you make when you rotate a ball with your palm on a flat surface. When you rotate an object in this way, the shape of the cursor changes to a spiral to indicate that a rotation takes place. The object may also be rotated around its axes using the dials: dial 1 (rotation around the X axis), dial 3 (rotation around the Y axis), and dial 5 (rotation around the Z axis).

Translations may also be performed on the object displayed. With the rightmost mouse button pressed you can translate an object by defining the translation vector with the mouse pointer. Dials 0, 2, and 4 offer an alternative to translating objects along their X, Y, and Z axes, respectively.

Slabbing is a feature available only for volumes and it consists of moving a plane parallel to the viewing plane and clipping the object displayed against it. Dial 7 allows you to move the clipping plane.
5.2.2. Recording Images and Producing Movies

The Record button in the command area allows the user to record the contents of the main drawing area in a file in TIFF format. The files are successively numbered: _Tframe10.tiff, _Tframe11.tiff, etc. (Note that if a file with one of these names already exists, an error occurs, so it is better if you remove any existing files with these names before you do the recording or change their names). When you press the Movie button (also in the command area), all files _Tframe*.tiff are concatenated into a movie file: _Tmovie.mvc2 in MVC2 format (an SGI video compression scheme with relatively fast decompression scheme) and the Silicon Graphics movieplayer is invoked. If a movie file with the name _Tmovie.mvc2 exists already, then no concatenation takes place, and the movieplayer is invoked with the old _Tmovie.mvc2 file as input.

5.3. Image Processing

Tonitza allows the user to read and display image files in SGI and TIFF formats. The regular visualization options are not available for files in these formats, but you may process these images by applying various image processing transformations to them. The images are not displayed in the main drawing area, but in a separate viewer. The viewer and the transformations are part of the Silicon Graphics Image Vision Library. All the transformations available are listed in a menu which is displayed when you press the right-most mouse button inside the viewer. They are: revert the contrast, histogram equalization, histogram scaling, image sharpening, image rotation, image blurring, image rotating, zooming, Sobel transformation, Laplace transformation, and compass N/W. The rest of this section contains a brief description of these techniques [GONZ87].

(a) Image Smoothing: edges and other sharp transitions (noise) in an image contribute heavily to the high-frequency content of its Fourier transform. Blurring is an image smoothing technique achieved by attenuating a specified range of high frequency components in the transform of a given image. (b) Arithmetic processing: arithmetic operators perform operations on each pixel of the image data. Inverting is an example of such an operator which takes the one's complement for every pixel in the data. (c) Histogram
Modification techniques for image enhancement: a histogram of the pixel value content provides a global description of the appearance of an image. Let the variable represent the values of the pixels in the image to be enhanced. The gray levels in an image are random quantities in the interval $[\min, \max]$. Their probabilities are given by the relation: $p(r_k) = \frac{n_k}{n}$, $\min \leq r_k \leq \max$, $k=0,1,...,L-1$, where $L$ is the number of pixel values, $p(r_k)$ is the probability of the $k^{th}$ value, $n_k$ is the number of times this value appears in the image, and $n$ is the total number of pixels in the image. A histogram is a plot of $p(r_k)$ versus $r_k$, and the technique for obtaining a uniform histogram is known as histogram equalization. Histogram scaling is another technique for image enhancement which clamps the pixel values to a specified percentage of the high and low intensity pixel and scales the remaining data between the clamp values. (d) Edge detection techniques: are image sharpening techniques for highlighting edges in an image. The most common sharpening techniques use gradient operators that produce edge-enhanced images by performing orthogonal convolutions with particular kernels. Such operators transform image data by computing a weighted of the pixels in the neighborhood surrounding the target pixel. The size of the neighborhood and the weights used for neighboring pixel values are defined by the kernel. The Sobel operator uses a bias value to be added as each pixelwise convolution is performed and an edge mode which shows how pixels at the edge of the image are to be handled. The Laplace operator is similar to the Sobel operator which allows you to select one of two predefined kernels. Compass operators measure gradients in a specified direction. North, or 0 degrees, is the top of the image.
6. IMPLEMENTATION NOTES

Tonitza version 2.0 consists of several modules which together sum up to approximately 54,000 lines of code. It is written in C and it includes calls to the X-Windows & Motif and OpenGL libraries. The skeleton of the Tonitza interface has been designed using the BX interface builder.

6.1. The OpenGL Library

The OpenGL graphics system is a software interface to graphics hardware (GL stands for Graphics Library). It allows the creation of interactive programs that produce color images of moving three-dimensional objects. It consists of approximately 120 distinct commands for specifying objects and operations needed to produce three-dimensional applications.

OpenGL is designed to work efficiently even if the computer that displays the graphics isn't the one that runs the graphics program. This is the case in a networked computer environment. The computer on which the program runs and issues OpenGL drawing commands is called the *client*, and the computer that receives those commands and performs the drawing is called the *server*.

OpenGL is designed as a hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead the windowing system that controls a particular hardware must be used. Similarly, OpenGL does not provide high-level commands for describing models of three-dimensional objects. The desired objects must be built up from a set of geometric primitives: points, lines, and polygons. More information about OpenGL can be found in [OPGL93].

6.2. The BX Interface Builder

Builder Xcessory (BX) is a tool for building and testing applications using OSF/Motif and the XtIntrinsics. The tool has two modes: **Build Mode** and **Play Mode**. In Build
Mode interface objects (called widgets) can be created and edited. The hierarchical structure of the interface is displayed in a **Browser**. The resources associated with a widget, such as geometry management, callbacks, colors, and fonts can be set using a **Resource Editor**. The appearance of the interface can be tested in Play Mode. **Note** that the user-defined callbacks are not connected.

Builder Xcessory can output C, C++, and OSF/Motif User Interface Language (UIL) code. It also reads UIL files. More information about BX and how to use it can be found in [BXUG93].

6.3. The Organization of the Program Modules in Tonitza

6.3.1. The Main Program

The file `toni.c` contains the main program:

- `main(argc, argv)`: routine containing the application event loop.

6.3.2. Creation Routines

All routines for creating the Tonitza widgets are contained in the file `creation.c`. The file contains entries of the form `Createform***()`. Each form corresponds to a top-level shell. These routines are called from the main program. **Note** that the file is relatively large (22,000 lines) and some of the routines may not be optimized at compile time, even if the `-O` option is used.

6.3.3. Callbacks

Some widgets provide hooks that allow the application to define procedures to be called when some widget-specific condition occurs. These hooks are known as **callback lists** and the procedures are known as **callback functions**, or simply **callbacks** [YOUN90]. The form of every callback is `void callbackProcedure(widget, client_data, call_data).

Event handlers are procedures invoked by the Intrinsics when a specific type of event occurs within a widget. The form of an event handler is `void handler(w, client_data,
Work procedures are a special type of callbacks that provide a limited form of background processing. They allow the programmer to register a callback and hide the details of the event processing. A work procedure is invoked by the Intrinsics whenever there are no events pending. The general form of a work procedure is `Boolean WorkProc(client_data)`. The procedure returns TRUE if the callback should be removed after it is called, and FALSE otherwise.

In Tonitza, all user-defined and BX generated callbacks are included in the file `callbacks.c`. This file also contains the event handlers and the work procedures.

6.3.4. Input/Output Routines

The files `input.c`, `filldsp.c`, `export.c`, and `detect.c` contain the I/O routines. They are:

- **input.c:**
  - `void parseArgs(argc, argv)`: for parsing of the command line arguments;
  - `void Usage()`: for printing the correct usage of the command line options;
  - `int readBrickHeader(file, format, length, na, nb, nc, nx, ny, nz, int_info, info)`: for reading the header of a file in brick format;
  - `int initShearMat()`: initialize the matrix for the shear transformation to be applied to the input volume (for non-orthogonal volumes);
  - `int readMapI2(fname)`: read a file in MAP INTEGER*2 format;
  - `int readSGIHeader(fname)`: read the header of a file in SGI format;
  - `int readTIFFHeader(fname)`: read the header of a file in TIFF format;
  - `void fillHeader()`: print the header information in a buffer whose contents will be displayed in the Header Info window.
  - `int readPlanes(file, length, na, nb, nc, firstp, lastp)`: read a file in PLANE_BY_PLANE format;
  - `int extractVolume()`: extract a subvolume from an input file for computing isosurfaces;
  - `int readDefaults()`: read the defaults file (if any) to restore the settings of a previous session.
filldsp.c:
- int filldsp(file, format, length, orientation, firstp, lastp, mode, head_addr, na, nb, nc, nx, ny, nz, plnsize, info) : read a file in BRICK format;
- int getall(miix, miyi, miiz, na, nb, nc, headlength, file_pointer, orientation, firstp, lastp, mode, plnsize) : internal routine called by filldsp().

export.c:
- exportSGIFrame(fname) : write to the disc an image in SGI format;
- compress() : RLE data compression scheme to be used with exportSGIFrame();
- exportTIFFrame(fname) : write to the disc an image in TIFF format.

detect.c:
- int detectDataFormat(filename) : detects the format of the input data file.

6.3.5. Drawing Routines

In Tonitza all drawing routines are contained in the file draw.c. They are as follows:
- int drawImage() : invoke a drawing routine corresponding to the current state of the program;
- int clearImage() : clear the main drawing area;
- int plotImage() : display two-dimensional representations;
- int drawHist() : display histogram;
- int drawAxes() : draw the axes of coordinates in the command area of the volumes window;
- int plot3D() : display three-dimensional representations;
- drawMaterial() : display an ellipsoid shaded using the current material in the material drawing area;
- drawCorrel(magF, correlC, re, gr, bi) : draw a plot of the correlation as a function of the magnification factor with small rectangles;
- drawAvgDens() : draw a plot of the average density as a function of the particle radius;
- drawRadCorr() : draw a plot of the correlation as a function of the particle radius;
- drawMagCorr() : same as drawCorrel() except lines are used instead of rectangles;
- `plotPoint()` : plot a point on one of the correlation graphs;

6.3.6. Routines for Generating Object Representations

All object representations generated in Tonitza are polygonal. The routines which implement these polygonalizations are `contour.c`, `isosurf.c`, and `sphere.c`.

**contour.c:**
- `void contour(grid, dim1, dim2, dx, dy, isolevel, list, size)` : build the contours corresponding to a given level by marching through all the grid cells in order;
- `void conInterp(isolevel, i, j, u, k, v, x, y, dx, dy)` : interpolate to find out the point where a given contour line crosses the edge of a grid cell;
- `int genContours(file, orient)` : generate contours at all requested levels;
- `int genStackContours(file, orient)` : generate a stack of contours.

**isosurf.c:**
- `int isosurj(grid, dim1, dim2, dim3, dx, dy, dz, prop, nlevel, ISOLEV)` : build an isosurface at a given level using the “marching cubes” algorithm;
- `void FindFaceEdges(dim1, dim2, dim3, ISOLEV, eindex, elist, A, B, C, D, dx, dy, dz)` : detect the edges of a polygon on a given face of a grid “cube”;
- `void interp(dim1, dim2, dim3, V, V, ISOLEV, x, y, z, TlX, ny, nz, dx, dy, dz)` : trilinear interpolation to find out where an isosurface intersects an edge of a grid cell;
- `int TracePoly(j, eindex, elist, h, j, k)` : determine the intersection polygon of the isosurface with a given parallelepiped;
- `int triangulate(prop, nlevel)` : generate a triangulation of an isosurface;
- `int genIsosurf()` : generate isosurfaces at all requested levels.

**sphere.c:**
- `int sphereToPoly()` : generate the polygonalization of a sphere;
- `int trilin(currFile, grid, j, i, k, x, y, z, val)` : perform trilinear interpolation;
- `int circleToPoly()` : generate the polygonalization of a circle.

6.3.7. Routines for Object Manipulation/Customization

The routines for manipulating and customizing the objects created by Tonitza are
grouped in the files `arcball.c`, `dials.c`, and `color.c`.

**arcball.c:**
- `void ballInit(ball)` : initialize the arcball rotation;
- `void ballPlace(ball, center, radius)` : change the arcball parameters;
- `void ballBeginDrag(ball, here)` : begin dragging;
- `void ballEndDrag(ball)` : finish dragging;
- `void worldToBall(ball, x, y, v)` : convert from world coordinates to arcball coordinates;
- `float getRotAngle(q)` : extract angle of rotation from quaternion;
- `void getRotAxis(q, v)` : extract axis of rotation from quaternion;
- `void quatMult(q1, q2, q)` : multiply two quaternions;
- `void quatInv(q1, q)` : calculate the inverse of a quaternion;
- `void vectToQuat(v, q)` : convert a vector to a quaternion;
- `void vectProd(v1, v2, v)` : calculate the vector product of two vectors;
- `void dotProd(v1, v2, s)` : calculate the dot product of two vectors.

**dials.c:**
- `void SetDialLabel(num, ptr)` : associate a label with a dial;
- `void GetDialState()` : retrieve the state of a dial;
- `void OpenDialsBox()` : initialize the dials;
- `void HandleDialEvent(ptr)` : handle a dial event.

**color.c:**
- `int initColor()` : initialize colormap variables;
- `int allocColors()` : allocate as many private colors as possible;
- `void computeColorCellDim()` : compute the color cell dimensions according to the number of colors allocated;
- `void drawColorCells()` : draw all color cells using the colors in the Color[] array;
- `void drawColorBars()` : draw all vertical color bars using the colors in the Color[] array;
- `void interpRed(i0, j0)` : interpolate the red component between knots i0, j0;
- `void interpGreen(i0, j0)` : interpolate the green component between knots i0, j0;
- void interpBlue(i0, j0): interpolate the blue component between knots i0, j0;
- void setColors(): update the RGB components of the colors in the colormap;
- void drawKnotMark(i, j): draw a small white square in the center of a color cell that has been selected;
- void placeKnotMark(i, r, g, b): update the knot arrays;
- void setKnots(): allocate and initialize the RGB knot arrays;
- void drawGraph(knots, col): draw red, green, blue polylines corresponding to the current configuration of knots;
- void drawSelectMark(k): draw a red "X" in the center of the currently selected color cell;
- void drawMarks(): draw black vertical lines for all cells that correspond to knots;
- void updateCol(): convert colors to floating point numbers between 0 and 1 to be used in the drawing routines;
- Pixel name_to_color(w, name): returns a Pixel data structure corresponding to a color name.

6.3.8. Computation Routines

The routines for correlation and histogram computation are contained in the files correl.c, sphere.c, and hist.c. They are:

correl.c:
- void extractSlabs(): determine the union of all correlation slabs;
- float cointerp(i, j, k): interpolate the value for grid point (i, j, k) from the magnified data set;
- int findCorrCoeff(v, sv, coef): determine the correlation coefficient of two data sets;
- int validAnnulus(val): check that a correlation annulus is properly being defined;
- float getSphereCorrel(r): compute the correlation of two data sets on a sphere of a given radius;
- float cointerpF(i, j, k): floating point version of cointerp();
- int computeScalingParams(): calculate the scaling parameters.
sphere.c:
- float getAvgSphere(radius) : compute average density on a sphere of a given radius.

hist.c:
- int findMinMax(min, max) : find min, max values for the current histogram property;
- int getFrequency(val) : find the frequency of a particular value in the data;
- int checkEntry(val) : check whether a data value has been used previously to compute its frequency;
- initHistogram() : initialize the histogram data structures;
- getInterval(val) : determine the interval to which a particular value belongs.

6.3.9. Routines for Defining the Fonts

The file font.c contains the routines for defining the font to be used in all OpenGL drawing areas. They are:

font.c:
- void makeRasterFont() : set the font;
- void printString(char *s) : print a string of characters using the current font.

6.3.10. Error Handling Routines

The file error.c contains the routines for error handling, displaying error and warning messages, and summaries. They are:

error.c:
- void popupMessage(w, code) : fill in and display the contents of the message buffer, according to the error/warning code;
- void popupDot(w, code) : fill in and display the contents of the dot message buffer, according to the message code (for "...") buttons);
- void popupSummary(w, code) : display information about the file and property to which current material is to be applied (summary for the material editor).
6.3.11. Utility Routines

There are two types of utility routines in Tonitza: those generated by BX and included in the file bxutil.c and those written by the authors and listed in the file util.c. The routines developed by the authors are:

**util.c:**

- int *swap*(ptr), *swap1*(ptr), *swap2*(ptr), *byte_rev*(i): byte swapping;
- int *sincos*(x, c, s): returns sin and cos of its first argument;
- int *clearAxes*(): set axes parameters to initial values;
- int *clearBox*(): set box parameters to initial values;
- int *clearState*(): set state parameters to initial values;
- int *clearSphere*(): set sphere parameters to initial values;
- int *clearPlane*(): set planar sections parameters to initial values;
- int *clearStack*(): set stack parameters to initial values;
- int *clearBall*(): set arcball parameters to initial values;
- int *clearReplicas*(): return to a single replica plot;
- int *clearVolume*(): set volume params to initial values;
- int *clearMatEditor*(): reset material editor widgets;
- int *ForceUpdate*(w): superset of XmUpdateDisplay that ensures that a window’s contents is visible before returning;
- int *sign*(x): determine the sign of its argument;
- int *setupNewFile*(ind): initialize state parameters for new file;
- float *minim*(f1, f2, f3, f4): compute the minimum of four numbers;
- float *maxim*(f1, f2, f3, f4): compute the maximum of four numbers;
- int *initFiles*(): initialize the files data structures;
- int *closeFile*(i): close an opened file;
- int *searchEmptyEntry*(): search for an empty entry in the file data structure, to allocate a new file;
- int *initCorrel*(): initialize the correlation parameters;
- int *rotateData*(): compute a rotated data set by interpolation.
APPENDIX

Error and warning messages may be displayed during a Tonitza session for a variety of reasons. The types of errors handled by Tonitza and the messages associated with each of them are described below.

<table>
<thead>
<tr>
<th>INTERNAL CODE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIDF (-1)</td>
<td>Error</td>
<td>Input file name or format are ill-defined.</td>
</tr>
<tr>
<td>IFNU (-3)</td>
<td>Error</td>
<td>Attempt to perform a computation or visualization task before an input file has been opened.</td>
</tr>
<tr>
<td>WDMU (-4)</td>
<td>Warning</td>
<td>Action has no effect when no object is displayed.</td>
</tr>
<tr>
<td>NEME (-17)</td>
<td>Error</td>
<td>Out of memory.</td>
</tr>
<tr>
<td>INFP (-16)</td>
<td>Error</td>
<td>The number of the first section in the input file (in BRICK format) is not valid.</td>
</tr>
<tr>
<td>TMPSR (-18)</td>
<td>Error</td>
<td>Maximum number of properties allowed has been exceeded.</td>
</tr>
<tr>
<td>MXMI (-23)</td>
<td>Error</td>
<td>Property range limits are invalid.</td>
</tr>
<tr>
<td>TMCO (-19)</td>
<td>Error</td>
<td>Too few or too many contour levels.</td>
</tr>
<tr>
<td>SSTB (-20)</td>
<td>Error</td>
<td>Stack badly defined: the first or last section numbers are invalid or it contains too many sections.</td>
</tr>
<tr>
<td>TMVE (-21)</td>
<td>Error</td>
<td>Out of memory when contouring.</td>
</tr>
<tr>
<td>IFSP (-5)</td>
<td>Error</td>
<td>Invalid number defining first sphere.</td>
</tr>
<tr>
<td>ILSP (-6)</td>
<td>Error</td>
<td>Invalid number defining last sphere.</td>
</tr>
<tr>
<td>ISST (-33)</td>
<td>Error</td>
<td>Sphere step is negative or zero.</td>
</tr>
<tr>
<td>INSP (-7)</td>
<td>Error</td>
<td>Maximum number of spherical sections allowed has been exceeded.</td>
</tr>
<tr>
<td>IMSP (-8)</td>
<td>Error</td>
<td>Out of memory when defining spherical sections.</td>
</tr>
<tr>
<td>INTERNAL CODE</td>
<td>TYPE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ICUB (-9)</td>
<td>Error</td>
<td>Data volume is not a cube.</td>
</tr>
<tr>
<td>WRES (-10)</td>
<td>Error</td>
<td>Sphere resolution too low: object may not approximate sphere accurately.</td>
</tr>
<tr>
<td>NEGS (-34)</td>
<td>Error</td>
<td>Sphere sampling is not positive.</td>
</tr>
<tr>
<td>UDPR (-22)</td>
<td>Error</td>
<td>Requested property does not exist.</td>
</tr>
<tr>
<td>HFUD (-11)</td>
<td>Error</td>
<td>Help file not available.</td>
</tr>
<tr>
<td>NOSP (-12)</td>
<td>Warning</td>
<td>Current drawing mode applies only to spherical sections.</td>
</tr>
<tr>
<td>NOCL (-13)</td>
<td>Warning</td>
<td>Action has no effect until drawing area is cleared.</td>
</tr>
<tr>
<td>UNKP (-14)</td>
<td>Warning</td>
<td>Unknown printer: image not printed.</td>
</tr>
<tr>
<td>INRL (-2)</td>
<td>Warning</td>
<td>Invalid number of replicas.</td>
</tr>
<tr>
<td>STOC (-24)</td>
<td>Warning</td>
<td>Stack available only for contours.</td>
</tr>
<tr>
<td>DFNI (-25)</td>
<td>Warning</td>
<td>Input data format not supported.</td>
</tr>
<tr>
<td>NOSE (-26)</td>
<td>Warning</td>
<td>Display mode not available for SGI and TIFF files.</td>
</tr>
<tr>
<td>NOGR (-27)</td>
<td>Warning</td>
<td>May not define grid properties for SGI and TIFF files.</td>
</tr>
<tr>
<td>MDCV (-28)</td>
<td>Error</td>
<td>Must define value for contour level.</td>
</tr>
<tr>
<td>HATS (-29)</td>
<td>Error</td>
<td>Property has too many distinct values: only continuous histogram available.</td>
</tr>
<tr>
<td>SAMP (-30)</td>
<td>Error</td>
<td>Histogram sampling too fine.</td>
</tr>
<tr>
<td>NSLD (-31)</td>
<td>Error</td>
<td>No space left on device.</td>
</tr>
<tr>
<td>COOF (-32)</td>
<td>Error</td>
<td>Cannot open output file.</td>
</tr>
<tr>
<td>IVDE (-35)</td>
<td>Error</td>
<td>Volume boundaries badly defined.</td>
</tr>
<tr>
<td>IVST (-36)</td>
<td>Error</td>
<td>Volume sampling is negative or zero.</td>
</tr>
<tr>
<td>ISNF (-38)</td>
<td>Warning</td>
<td>Isosurface not found.</td>
</tr>
<tr>
<td>ALTW (-39)</td>
<td>Error</td>
<td>Option requires two input files.</td>
</tr>
<tr>
<td>TMNF (-41)</td>
<td>Error</td>
<td>Maximum number of input files that can be opened simultaneously has been exceeded.</td>
</tr>
<tr>
<td>INTERNAL CODE</td>
<td>TYPE</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>FAEX (-43)</td>
<td>Warning</td>
<td>File has already been opened.</td>
</tr>
<tr>
<td>TFEF (-44)</td>
<td>Error</td>
<td>No input data files have been opened.</td>
</tr>
<tr>
<td>SREZ (-45)</td>
<td>Error</td>
<td>Sampling step (isosurfaces) is zero.</td>
</tr>
<tr>
<td>NRWS (-46)</td>
<td>Warning</td>
<td>Spherical sections not available for rotated maps.</td>
</tr>
<tr>
<td>WMBO (-47)</td>
<td>Error</td>
<td>Magnification range badly defined.</td>
</tr>
<tr>
<td>WMST (-48)</td>
<td>Error</td>
<td>Bad magnification step.</td>
</tr>
<tr>
<td>XILI (-49)</td>
<td>Error</td>
<td>X correlation slab badly defined.</td>
</tr>
<tr>
<td>YILI (-50)</td>
<td>Error</td>
<td>Y correlation slab badly defined.</td>
</tr>
<tr>
<td>ZILI (-51)</td>
<td>Error</td>
<td>Z correlation slab badly defined.</td>
</tr>
<tr>
<td>IANN (-52)</td>
<td>Error</td>
<td>Invalid number of correlation annuli.</td>
</tr>
<tr>
<td>IANS (-53)</td>
<td>Error</td>
<td>Badly defined correlation annulus.</td>
</tr>
<tr>
<td>TMMG (-54)</td>
<td>Error</td>
<td>Magnification sampling too fine.</td>
</tr>
<tr>
<td>PRER (-55)</td>
<td>Error</td>
<td>Cannot record image file.</td>
</tr>
<tr>
<td>WROF (-56)</td>
<td>Error</td>
<td>Output file not in MAPI2 format.</td>
</tr>
<tr>
<td>CCOF (-57)</td>
<td>Error</td>
<td>Failed to write file to disc.</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

We would like to thank Timothy S. Baker and Jodi K. Muckelbauer for providing the data sets for the Ross-River and the Coxackievirus B3 used as examples in this document. Tim’s advice and criticism were invaluable to us. Michael Rossmann’s enthusiasm and encyclopedic knowledge have immensely contributed to our understanding of some of the computational problems arising in Structural Biology. We also thank Zhongyun Zhang for helping with the input data conversion, Holland Cheng, Paul Chipman, Steve Walker, and Norm Olson for their critical observations.

This research has been partially supported by the NSF grants BIR-9301210 and MCB-9527131, by a grant from the Intel Corporation, and by SIO, Scalable IO Initiative.
BIBLIOGRAPHY

[BAKE95] T. S. Baker, private communications and class notes.


