Practical Data Visualization Course – Assignment Writing Guidelines

1. Introduction

The goal of this document is to provide detailed support for the execution of the assignment coming with the Practical Data Visualization graduate course. Before starting with the assignment (after the lecture series is over), read this document in detail. This will ensure that you know

- what type of problem you should select for the assignment
- which are all aspects that you have to cover in the assignment (and report)
- the order in which you should cover these aspects (steps of the assignment)

The steps of the assignment are: problem description, data description, visualization design, visualization implementation, and description of findings. They are detailed below.

At the end of executing these steps, you should have produced a *report* of 20-25 pages, containing a detailed coverage of the execution of your assignment, complemented by a number of images (snapshots of your visualizations and, possibly, also schematic diagrams describing your design).

2. Problem description (step 1/5)

The assignment starts by selecting a *problem* whose solution will be supported by one or more visualizations. The optimal way to do this is to consider your own research (or the research of a colleague or scientist with whom you interacted). This ensures that the problem you will next treat is relevant to you; you know the problem domain; and, by doing the assignment, you will learn insights which will prove directly useful and usable in your direct work.

A problem can be of many types and extents. For example, it can be formulated as a task or goal whose resolution is helped by visualizations. Instances of this case are the following examples:

Given a matrix whose elements change in time, show the change by highlighting patterns such as periodic oscillations, plateaus, slowly vs rapidly changing values, blocks of values that change in sync, and stable/constant values.

Given a set of large data matrices, show which subparts of these matrices are to be found in several matrices in the collection.

Given a network-like structure consisting of a large number of hubs and links, show the core, or skeleton, of this structure.

Given a complex electrical circuit, show how current and voltage values flow (change) over its components in time.

Given several thousands of experimental measurements of the same phenomenon, each consisting of tens or more of measured variables, find which groups of measurements are strongly similar.

Given several thousands of experimental measurements of the same phenomenon, each consisting of tens or more of measured variables, find which groups of variables are strongly correlated, respectively inversely correlated, in time.

However, you can formulate a problem also in terms of a question that your visualizations should next aim to solve. For example:

Given several tens or hundreds of time series, each having hundreds of measurements (samples), how to show the most similar time series?

We have a 2D image, acquired with a microscope. For each image pixel, besides its color, we also know the measurement precision or uncertainty (which can vary largely for different pixels). How to intuitively show to a scientist both the pixel colors and the measurement uncertainties they were acquired with?

We have a very long (DNA-like) sequence of ordered elements, where each element has a type taking values in a small collection of 10 types (e.g., is of type A, B, C..., J). How to show repeating structures (of variable lengths) along this sequence, if the sequence has 200000 elements?

We have a table of values. Each value is, however, not a single number, but a distribution. How to show this table so that we recognize similar distributions easily?

Guidelines for problem selection:

- don't make the problem too complicated; it will take too much effort or technical skills to solve;
- don't make the problem too simple or not related, in any way, to visualization; it will not make sense to solve that problem;
- the problem can be a sub-part of a more complex problem in your research; scope it so that its size and level of detail is suitable;
- make the questions or tasks related to your problem as specific as possible! If not, then it is hard, even impossible, to justify next that the proposed solution is indeed a (good) solution for that problem.

Related reading: Module 1

3. Data description (step 2/5)

In this step, you describe the problem in terms of datasets and related operations. In other words: you abstract from the problem universe (e.g. volts, amperes, molecules, bacteria, chemical samples, reactivity, biological diversity, ...) to a mathematical universe, modeled by the *dataset* notion.

To do this, you must try to express the data that is involved in your problem and the tasks that address that data in terms of

- type of *dataset* (table, grid, 2D/3D shapes, time series, set, sequence, tree, graph, ...)
- type of *attributes* (quantitative, integral, ordinal, categorical, text, relations, ...)
- type of *operations* you can do on the attributes (see the operations allowed by the above attribute types)

- type of *measurements* you want to do on the data (e.g. find similar items, sort items on a criterion, highlight outlier values, highlight trends/patterns, find correlations or lack thereof, ...)
- *size* of your dataset (number of observations, number of variables)

The aim of this reduction to a mathematical framework is to next allow you to reason about which visualization techniques are (best) applicable to your context. For this, you will compare the dataset that fits your problem with the dataset requirements exhibited by the various visualization algorithms presented in Module 2.

Several observations are important in this process:

- your problem may not be captured by a single dataset, but a collection of datasets of several kinds. This is fine, as long as you can next find a good visualization for each such dataset;
- for your dataset, there will be (generally) several visualization algorithms applicable. To further select among them, consider other aspects such as dataset size, availability (and ease of use) of the respective algorithms, and the types of tasks they support *vs* the tasks you have for your data;
- even after the above algorithm selection, you may be left without a single grand winner (in terms of visualization algorithms); this is not a problem – you may propose a solution in which you use e.g. two visualization algorithms to highlight different aspects of the same dataset (the 'multiple linked views' metaphor).

Example: Consider the first problem sketched in Sec. 1:

Given a matrix whose elements change in time, show the change by highlighting patterns such as periodic oscillations, plateaus, slowly vs rapidly changing values, blocks of values that change in sync, and stable/constant values.

We can encode the data of this problem as a set of 2D matrices, or tables, where each table gives the data samples at a moment in time. Alternatively, we could encode the data as a 3D 'data cube' obtained by stacking these 2D matrices in their temporal order. Alternatively, we could think of a 'generalized matrix' of the size of our input matrix, where each cell is not just a number, but the entire evolution (time series) describing the variations of that item in our time-dependent matrix.

Related reading: Module 2

4. Visualization design (step 3/5)

In this step, you describe how the dataset and tasks found in step 2 are further mapped to visual variables. In other words, you describe

- how you encode your data into elements such as shapes, sizes, colors, order of elements, layouts; and
- how you next propose to address the tasks found in step 2 by visually inspecting the produced visualization(s).

To do this, consider (again) the visualization algorithms that best match the type of dataset you have and the tasks/questions you have. Also, consider the attribute types in your dataset – recall, as discussed in Modules 2 and 3, that not all attribute types map equally well to all visual variables. Finally, consider the data size: some visual encodings work well for small datasets only (e.g. graph drawings), while others work very well for large datasets (e.g. table lenses or timelines). Once you have found a design (or several complementary or alternative designs), describe and defend your choices in detail, including their known limitations.

Example: Consider our running example problem. Let's say we choose, as dataset model, the 'generalized matrix' idea (see Sec. 3). We then could propose a small-multiple design: We draw the matrix as a grid, and within each grid we draw a small timeline, or graph, showing the evolution of values of that cell. This should scale relatively well to matrix sizes up to about 20x20. Next, we can use encoding redundancy to highlight certain data patterns, such as local maxima, outliers, or trends – see the redundant encoding in bar length and bar coloring in Module 2 for the table lens example. Using this design is simple: the location of graphs in cells tells us at which specific cell we are looking; by visually comparing these graphs, we can find e.g. which ones are highly similar, which ones exhibit outliers, and so on. One challenge here is that the design doesn't scale too well – already comparing 20x20=400 graphs visually is very hard. Another challenge is data normalization: By normalizing all graphs to the same size (of a matrix cell), we loose insight in the absolute values (scale) of the phenomena. For example, a voltage-graph on the scale of millivolts will look in this design identical to a voltage-graph having the same trend but amplified thousand times (on the scale of volts). To fix this, we may actually use the color to code the absolute data values.

Related reading: Modules 2, 3 and 4.

5. Visualization implementation (step 4/5)

In this step, you take your proposed design and realize (implement) it using one or more visualization tools. To help with tool selection, we provide a list of many well-known visualization tools at the end of this document. Most these tools are freely available, and several of them can be (easily) used with little or even no programming experience.

Since each of you is free to come with their own visualization problem, dataset, and tasks, and most of you have different backgrounds, it is virtually impossible to come with a single tool offer that will fit you all. As such, *tool selection* is a critical step to do.

Tool selection is a very challenging task: Finding a suitable tool depends on a large number of factors: your programming experience; licensing issues; platform you need to run the tool on; size and type of your dataset (not all tools support all types and/or sizes of datasets); kind of visualization you want to generate; freedom you need to configure visualization parameters (some tools allow configuring everything; other tools allow only a limited subset of options); and, last but not least, time you have to invest in the entire tool-searching, tool-testing, and tool-learning processes. However, doing this search is a very good learning experience: During this, you will learn about new tools (maybe not useful today, but really useful tomorrow); will see what state-of-the-art visualization tools typically offer to, and require from, the user (and thus may decide that next you really need to spend time to learn that scripting language or data format); and will be exposed to visualization techniques not discussed in the lecture (and may find accidentally solutions to other of your long-standing problems).

When searching for a suitable tool, the following points are important:

- be *critical:* Do not spend too much time testing each single tool. Rather, move quickly through the tool descriptions provided at the end of this document, focus on a subset of tools, and then visit them in order, download them, and decide quickly if they have a chance to fit your context. Eliminate first, so you're left quickly with a small set of good candidates.

- be *specific:* a tool may indeed feature a certain technique (e.g. 'drawing graphs'), but is that precisely the kind of technique you need? (e.g. 'drawing graphs of about 1000 nodes, with several attributes per node, that I want to easily encode into node color, size, shape, and labels, and I want to do all this relatively quickly, and I don't know how to code in C++?')
- be *selective*: you may find that no tool does it all, but several tools provide partial solutions to your problem. This is fine – use then each tool to get a bit of the puzzle solved. Also, remember, this course is about visualization design, not software-tool design. As such, it is absolutely no problem to e.g. manually perform a number of operations to construct your final visualization (e.g. adding labels, annotations, or color legends to plots, assembling/arranging plots into a final visual design). Examples of this way of working are presented in Modules 3 and 4.
- *optimize*: start by finding a solution to your simplest problem part. You will then arguably find many tools that do it. This is good, since you're quickly done with that part. Then you have more time to focus on the complicated bits.

Example: Consider our running example problem. Let's say we choose the 'small multiples' visual design outlined in Sec. 4. To implement this, we can use a multitude of tools, such as e.g. Matlab, Mathematica, ManyEyes, Prefuse, or similar (see appendix). We can then use any of these tools to generate the data plot for each matrix cell – of course, carefully tuning the visual encoding and visual attributes to best map the data. Next, we can assemble the plots in a matrix layout, and optionally add annotations such as labels and selections, by using e.g. Illustrator, PowerPoint, or any similar graphics editor.

Related reading: Module 4.

6. Description of findings (step 5/5)

In this step, you describe how the visualizations realized in step 4 solve the original problem – that is, complete the tasks and/or answer the questions distilled in step 1. To do this, you need to actually do a bit of 'storytelling': This involves walking the path from the *images* (in the visualization) to the original *problem domain* – or, in other words, explain the 'inverse mapping' you propose (see Module 1). This may sound complex, but is actually very easy if your visualizations are good: You only need to explain how you go from seeing some interesting patterns in the visualizations to mapping these patterns to data and finally to findings related to your problem.

Several points are important here:

- do not try to create a *single* snapshot or visualization that 'tells it all'. Unless your problem is trivial and the dataset is small, this is virtually impossible. You will likely need several such snapshots, each describing one or a few different findings. When putting the insight from all these findings together, we solve the problem.
- try working *iteratively*: Generate, for example, a snapshot that gives some hint or general idea about where our answer is. Describe how you see this in the snapshot. Then, we get to the idea that we need to dig deeper in a subset of the data (which we saw as being interesting in the overview). Insert then a new snapshot, showing that subset. Repeat the process (describe what you see in this 2nd snapshot, etc). Iterate the process until you have found all answers pertaining to your original questions.

- be generous with *annotations*: A bare snapshot may require half a page of explanations which may not be easy to give, since it's not easy to refer in the text to e.g. 'that peak in the top-right of the image being similar to that valley behind the descending slope half-way the bottom part of the image'. Just add annotations to the image to which you can next refer in the text. This makes the textual explanation much clearer *and* also much shorter. See the examples provided in Modules 3, 4, and 5.
- be *fair*: If your visualization really shows some finding, good, say it. If it doesn't, do not artificially increase its added-value by stating it helps you find things that it really doesn't. Rather, state its limitations (with respect to your questions/tasks) explicitly. Even better: if you know how these limitations could be overcome with reasonable effort (e.g. using a better visualization tool that allows you to configure that parameter X or Y, or using a visualization discussed in the course for which you couldn't get a tool running on your data), state this explicitly. This means you have successfully found a solution – you just miss an implementation thereof.

Example: Consider our running example problem. Let's say we implement the 'small multiple' solution using the TableVision tool (see Module 2). We then could use bar length and bar color to highlight data values. A typical 'inverse mapping' explanation would then be: "In cells (12,13) and (3,1) of the matrix, we see two very similar graphs in terms of shape. This means that these two signals show the same trend, so they are strongly correlated. Additionally, we see that all cells in the rightmost column have almost flat graphs – which indicates that these signals are nearly constant for our simulation duration. As such, we will switch them off the final visualization (see next Figure), since they don't bring any additional insights. We also see that the diagonal cells (1,1) up to (5,5) exhibit one outlier, visible in the shape of the narrow, red, peak in these graphs. However, we see that this outlier is shifted at different positions in time. This indicates that the 'power surge' that our experiment was about is not felt at the same precise moment at all our grid stations".

Related reading: Modules 3, 4 and 5.

Additional material

To support your assignment execution, the additional material below is useful:

- Slides of the Practical Data Visualization lecture. These are available online at

  http://www.cs.rug.nl/svcg/PracticalDataVis

  Read the slides carefully. They contain additional material (besides the one discussed during the lectures), such as visualization algorithms and examples of visualization applications. Also, they contain many pointers to articles, visualization tools (available for free to download), and datasets (available for free to download)

- List of visualization tools (from the book *Data Visualization – Principles and Practice* (A. C. Telea), 2nd edition, CRS Press, 2014). This gives a comprehensive list of visualization tools for scientific/spatial data, non-spatial/abstract data, image data, shapes data, and more. Be sue you study this list to find the candidate tools for your assignment.

Appendix

Visualization Software

One important element of the five-dimensional classification model for visualizations presented in Section 4.3 was the *medium*, or type of drawing canvas that the rendering takes place on. There are many examples of early visualizations that use paper as the medium [Spence 07]. Modern architectural blueprints can also be seen as visualizations that use a printed medium. Yet, by far the largest class of visualization applications described in this book have one thing in common: they use the computer screen as a medium.

Using the computer to do data visualization is a natural choice from several points of view. First, many visualization scenarios are, by their very nature, *explorative*. This makes interactive visualization tools the best instruments for such cases. Second, the datasets to visualize usually come in electronic form. Third, the large amounts of data, or dynamically changing datasets, make computer-based visualization tools again the natural choice.

In this appendix, we provide an overview of a number of issues concerning visualization software. First, we discuss how visualization software can be classified from an architectural perspective (Section A.1). Next, we provide a list of several representative visualization systems for scientific, imaging, and information visualization data, in order to illustrate the various flavors of systems available to practitioners.

A.1 Taxonomies of Visualization Systems

Visualization software tools are central to creating successful visualizations. Besides the provided functionality, such systems have also to cover several nonfunctional requirements in order to be effective. Relevant attributes in the latter group include the following:

- Efficiency: The software should produce visualizations quickly. This can mean minutes for some applications, but fractions of a second for others, such as interactive applications.
- Scalability: The software should be able to handle large datasets within given performance and resource bounds.
- Ease of use: The software should provide an easy-to-learn-and-use interface for its intended user group.
- Customizability: The software should allow a simple and effective way to customize it for specific tasks, scenarios, problems, or datasets.
- Availability: The software should be available to its intended user group under specific conditions (e.g., license and platform).

Modern visualization applications are complex software systems containing tens or hundreds of thousands of lines of code organized on several layers. With respect to this layered application architecture, the users can have different *roles*. One such classification identifies three roles: end users, application designers, and component developers [Ribarsky et al. 94]. End users are the final customers of a visualization application, and use it to obtain insight into a given dataset, typically by means of customized user interfaces that support domain-specific tasks. Such applications are also known as *turnkey systems*. Application designers construct turnkey systems for the end users, typically by assembling a set of premade software components and providing them with the needed configuration and user-interface elements. Finally, component developers program software components that implement visualization algorithms and datasets, and provide these to application designers as ready-made visualization software packages or libraries.

From this perspective, visualization software can be classified into three classes: libraries, application frameworks, and turnkey systems. Libraries provide application programmer interfaces (APIs) that contain the data types and operations that constitute the basic building blocks of a visualization application, such as the ones presented in Chapter 3. At the other end of the spectrum, turnkey systems provide custom user interfaces, presets, and configurations designed to support specific tasks. Application frameworks fall between these two extremes. They encode a set of fixed domain-specific rules, operations, and functions as a backbone to which an open set of components can be added. The components use the backbone to interact and provide functionality at a higher level, and in a more compact way, than bare libraries. Several mechanisms exist for adding components to the framework and composing them in. A popular design metaphor presents the components to the application designer in a visual, iconic form. Applications are constructed by interactively assembling these iconic component representations. This allows nonprogrammers to quickly and easily prototype new applications without programming. The AVS, ParaView, and VISSION applications illustrated in Figures 4.6, 4.9, and 4.8 in Chapter 4 are examples of application frameworks.

In the next sections, we give several examples of visualization software systems used in practice. Instead of using an architectural taxonomy into libraries, frameworks, and turnkey systems, we have opted for a domain-centered taxonomy into three classes: general scientific visualization systems (Section A.2), medical and imaging systems (Section A.3), and information visualization systems (Section A.5). Given the size of the field and the rapid rate at which new software is produced, the list of systems presented here is definitely not exhaustive and limited in choice. However, we believe that this list can serve as a useful starting point for the interested reader in search for a given visualization software tool or component.

A.2 Scientific Visualization Software

The systems listed in this section fall in the category of general-purpose *scientific visualization* software. The target of such systems is primarily the visualization of datasets defined as two- and three-dimensional grids of various types with scalar and vector attributes, such as created by scientific simulations or data-acquisition processes. The main application domains targeted are engineering, mechanics (both in research and in the industry), and weather and geosciences. However, some of the systems provide also support for medical imaging, tensor visualization, and information visualization.

The Visualization Toolkit (VTK)

Type:	Class library (written in C++)
Availability:	Open source
Address:	http://www.kitware.com/vtk/

Description:	VTK is a set of class libraries written in C++. Classes come in two main flavors: datasets and algorithms. Dataset classes range from low-level containers, such as lists and arrays, to full-fledged uniform, rectilinear, structured, and unstructured grids. Several hundred algorithm classes provide grid manipulation, slicing, interpolation, contouring, streamlines, image processing, polygonal and volume rendering, and information visualization techniques for graphs and table datasets. VTK is arguably one of the leading data visualization libraries at the moment.
Utilization:	Applications are built by assembling dataset and algorithm class instances into a pipeline. This is done either via the native compiled C++ API or its wrappings in interpreted languages, such as Python, Java, Tcl, and recently also .NET. The basic VTK building blocks offer a wide functionality. Yet, constructing a complete visualization application, and even more so extending VTK with one's own algorithms, requires a fair amount of programming effort and knowledge of the VTK API and its programming paradigms.

MeVisLab

Type:	Turnkey system/application framework
Availability:	Restricted open source (written in C++)
Address:	www.mevislab.de/
Description:	The MeVisLab system can be used as an end-user tool for processing and visualizing scientific datasets. Its main operation mode is very similar to AVS/Express, IRIS Explorer, and SCIRun (described below). A visual editor allows assembling a dataflow network from existing components available in a number of application-specific libraries. Component parameters can be controlled by customized GUIs or by direct interaction in the available 2D and 3D viewers. One of the main strengths of MeVisLab is the huge number of available components, provided by the core software itself, third-party community developers, but also the integration of VTK, ITK, and Open Inventor visualization and graphics libraries within a single framework. As such, MeVisLab's functionality covers extensively scientific visualization, volume/medical visualization, and image processing. Several components, such as volume processing and volume rendering, benefit from optimized GPU implementations.
Utilization:	MeVisLab operates mainly as an end-user system. The application design freedom is slightly smaller than in systems such as AVS/Express or IRIS Explorer, but higher than in ParaView or MayaVi. Carefully designed user interfaces and a comprehensive documentation make the system easy to learn and use, and especially suited for educational, rapid data exploration, or demonstration scenarios. Modules can be added via

a plug-in mechanism, and are actively contributed by the open source community. MeVisLab comes with mainly two license types. The free license offers unrestricted use for research and academic purposes, but a restricted set of features. The paid license is further split into a full-feature variant for nonprofit organizations, and a full-feature variant for commercial usage.

AVS/Express

Type:	Application framework
Availability:	Commercial
Address:	www.avs.com/solutions/express/
Description:	AVS/Express is an application framework for development of high-end scientific visualization solutions. AVS/Express provides more than 800 visualization building bricks (algorithms) that cover scalar, vector, and tensor visualization on several types of grids, much like VTK. Several extensions of AVS/Express provide support for parallel processing and high-end virtual-reality visualizations. Some extensions also target information visualization (OpenViz toolkit), operational industrial monitoring (the PowerViz toolkit), and scientific presentation graphics (the Gsharp toolkit). A snapshot of the AVS tool in action is shown in Figure 4.8.
Utilization:	Applications are built by assembling premade components into a pipeline, somewhat similar to the VTK paradigm. This can be done programmatically in C or C++, or interactively, via a point-and-click visual application designer. The visual editor allows rapid application prototyping, user interface construction, and application steering. The AVS/Express components are less fine-grained than VTK classes, for example.

IRIS Explorer

Type:	Application framework
Availability:	Commercial
Address:	http://www.nag.co.uk/welcome_iec.asp
Description:	IRIS Explorer is an application framework for development of high-end scientific visualization solutions. Its user group and philosophy is quite similar to AVS/Express. The provided visualization functionality covers application domains as varied as life sciences, chemistry, medical imaging, geology, financial modeling, and aerospace engineering. IRIS Explorer builds its versatility on top of several major software components, such as the well-known Open Inventor 3D and ImageVision

graphics libraries and the Numerical Algorithms Group (NAG) numerical libraries. IRIS Explorer is particularly attractive for users who wish to combine numerical simulation code with visualization facilities.

Utilization: IRIS Explorer provides a visual application builder based on a dataflow application architecture, much like AVS/Express. Modules can be developed in C, C++, and FORTRAN, but also in a proprietary scripted language called SHAPE, which offers an easier way to manipulate complex n-dimensional datasets. Modules are next visually assembled in so-called maps, which are essentially dataflow networks.

SCIRun

Type:	Application framework
Availability:	Open source
Address:	http://software.sci.utah.edu/scirun.html
Description:	SCIRun is an application framework developed for the creation of scientific visualization applications. SCIRun is very similar in aim and scope to AVS/Express. It provides a set of modules for scientific data visualization that can be connected into so-called networks, following a dataflow application architecture. SCIRun has been used to construct visualization applications for several domains such as finite element numerical simulations, (bio)medical imaging, and computational steering.
Utilization:	Similar to AVS/Express, SCIRun allows applications to be constructed visually, by editing the dataflow network, or programmatically. Once constructed, applications can be packaged into so-called PowerApps. These are dataflow networks provided with custom user interfaces into turnkey applications that facilitate specific exploration scenarios. Several such PowerApps are available for different domains, such as segmentation (Seg3D), tensor visualization (BioTensor), volume visualization (BioImage), and finite element problems (BioFEM). All in all, SCIRun is a mature environment that covers most needs and requirements of the users of a scientific visualization framework.

ParaView

Type:	Turnkey system/application framework
Availability:	Open source
Address:	http://www.paraview.org/
Description:	The ParaView system can be used as an end-user tool for visualizing scientific datasets. The main operations for data filtering, selection, mapping, and rendering are provided, such as slicing, isosurfaces,

	streamlines, and interactive viewing. ParaView provides an intuitive and simple graphical user interface that allows one to both prototype a visualization application and set its various parameters interactively. ParaView is built on top of the VTK class library. The user interface layer is written in the Tcl/Tk scripted languages. Most of the illustrations used in this book were created using ParaView, unless otherwise specified in the text.
Utilization:	ParaView operates mainly as an end-user system. The application design freedom is considerably less involved, but also easier to learn, than AVS/Express, for example. ParaView makes an excellent system for learning the basics of scientific visualization without having to be a programmer. Only the core VTK functionality is exposed in the user interface. However, developers can add new modules to the ParaView user interface using a mix of Tcl, C++, and XML wrappers.

MayaVi

Type:	Turnkey system/application framework
Availability:	Open source
Address:	http://mayavi.sourceforge.net/
Description:	The MayaVi system is an end-user tool for visualizing scientific datasets. Its architecture, provided functionality, and intended user group are very similar to ParaView's. MayaVi is also built as an interactive front-end on top of the VTK class library. However, in contrast to ParaView, MayaVi uses Python as a scripted (interpreted) language to bind user interface functions to the compiled C++ libraries.
Utilization:	MayaVi operates mainly as an end-user system, similar to ParaView. The user interface is structured differently. A relatively greater emphasis is put on executing operations by writing Python commands at a user prompt than via the user interface itself. The user interface exposes more of the VTK implementation details and is relatively lower-level than the one of ParaView. Overall, although MayaVi and ParaView are quite similar in intention, we find ParaView easier to learn and use and more mature than MayaVi.

A.3 Imaging Software

In this section, we list a number of imaging software systems. By "imaging," we refer to several functionalities related to the manipulation and visualization of 2D and 3D image datasets. Such datasets occur frequently in medical practice as the output of the various scanning technologies, such as computed tomography (CT) and magnetic resonance imaging (MRI). Image datasets can contain scalar, vector, and tensor data. Imaging operations cover a wide range of tasks, such as basic data handling and manipulation, basic image processing, image segmentation and registration, shape recognition, image visualization, and volume rendering. Just as for the other types of software systems listed in this appendix, it is not possible to cover all aspects and variants of such systems, so we limit ourselves to a small selection of representative systems.

The Insight Toolkit (ITK)

Type:	Class library (written in C++)
Availability:	Open source
Address:	http://www.itk.org/
Description:	ITK is a set of class libraries written in C++. The main functionality targeted by the ITK toolkit is divided into three categories: image processing, segmentation, and registration. As such, ITK does not provide visualization (rendering) and user interface facilities. However, ITK can be combined with other toolkits, such as VTK, in order to construct complete visualization applications.
Utilization:	The software structure of ITK bears a number of similarities to VTK, which is not surprising given the fact that a large number of common organizations and people have been jointly involved in the development of both toolkits. ITK comes with a considerable amount of documentation in the form of books, courses, online material, examples, and demos. Also, the source code of several medical imaging applications built on top of ITK is available for downloading from the ITK site. However, just as for its older cousin VTK, using ITK to construct an imaging application requires a nonnegligible amount of effort, given the sheer size and complexity of the toolkit APIs.

3D Slicer

Type:	Turnkey system/application framework
Availability:	Open source
Address:	http://www.slicer.org/
Description:	3D Slicer is a freely available, open-source application framework for visualization, registration, segmentation, and quantification of medical data. 3D Slicer supports a wide array of tasks, ranging from the investigation and segmentation of volumetric CT and MRI datasets to

providing the basic mechanisms for more complex applications, such as guiding biopsies and craniotomies in the operating room and diagnostic visualizations. 3D Slicer can handle scalar, vector, and tensor data attributes. In particular, several functions for visualizing diffusion tensor images, such as principal component analysis, tensor color coding, tensor glyphs, and hyperstreamlines are supported. 3D Slicer is supported by a large number of organizations, and is used in a large number of research projects, as well as in clinical studies and actual applications in the medical practice. Several images created with the 3D Slicer tool are shown in Section 7.6.

Utilization: 3D Slicer has an architecture consisting of an end-user front end and a framework that manages a set of application libraries. The front end provides user interfaces and direct mouse-based manipulation of the data, such as picking, probing, and interactive streamline seed placement. The application framework allows one to add new plug-ins in order to provide custom-developed functionality. The 3D Slicer architecture was designed to facilitate adding a wide variety of plug-ins, ranging from standalone binaries (executables and shared libraries) to modules based on the VTK and ITK toolkits and even shell, Tcl, and Python scripts. Although this makes learning the software architecture of 3D Slicer more complex than that of other toolkits such as VTK or ITK, it also makes 3D Slicer more flexible in interfacing with a broad spectrum of third-party software components.

Teem

Type: Turnkey/libraries

Availability: Open source (written in C)

Address: http://teem.sourceforge.net/

Description: Teem is a set of coordinated libraries representing, processing, and visualizing scientific raster data. Teem provides functions that support a wide range of operations on n-dimensional raster data (uniform grids) of m-dimensional attributes. The functions of Teem are provided as a set of standalone executables designed much like UNIX filters, which are parameterized by command-line options. The generic data model of Teem, together with the modular decomposition of operations in terms of several filters, allows many complex operations on image volumes to be specified easily and compactly. The set of basic operations provided by Teem include convolution, slicing, resampling, interpolation, statistics, principal component analysis for tensors, color mapping, volume rendering, and tensor glyph visualizations. Several images created using the Teem software are shown in Section 7.5.

Utilization:	Data-processing and visualization tasks are typically written as shell scripts that construct and execute a dataflow pipeline by cascading the Teem basic filters. This is the easiest and most rapid way to use Teem to produce visualizations. If desired, Teem can be also used in terms of libraries providing APIs. Teem is not end-user software. As such, it does not provide user interface or interaction functions present in complete visualization applications. However, its generic, modular, and coherent API design allow such applications to be built on top of it. For example, the SCIRun and 3D Slicer applications integrate the functionality of Teem to provide high-level imaging capabilities.

ImageJ

Type:	Turnkey
Availability:	Open source (written in Java)
Address:	http://rsbweb.nih.gov/ij/
Description:	ImageJ is an application conceived for the easy processing of 2D images. Inspired by classical 2D image manipulation programs, ImageJ offers a wide set of basic image processing operations, such as contrast enhancement, smoothing, sharpening, denoising, edge detection; advanced image processing such as segmentation and manipulating stacks of 2D images, image sequences, or separate channels of the same image; and computing various image statistics and measurements, such as histograms and image calibration. Written in Java, ImageJ is optimized for the efficient processing of large images, using multithreading.
Utilization:	In typical end-user mode, ImageJ offers its processing capabilities via GUIs and a range of direct selection and manipulation tools. Basic functionality is extendable via a relatively easy-to-use plug-in Java interface. As such, hundreds of third-party plug-ins has been developed for ImageJ for both general-purpose image processing but also for specialized manipulation of various types of microscopy, biology, and medical images. Basic operations can be composed in so-called macros by using a built-in scripting language. Given the rich set of available plug-ins, ImageJ is a very competitive alternative to MATLAB for 2D image processing.

Binvox

Type:	Turnkey utilities
Availability:	Open source (written in C/C++)
Address:	http://www.cs.princeton.edu/~min/binvox

Description:	Binvox is a set of command-line utilities for the conversion of 3D polygonal meshes to volumetric formats. The main utility of this toolset is *binvox*, a program for converting polygonal meshes in a variety of formats (such as Stereo Lithography (STL), Stanford Polygonal File Format (PLY), Object File Format (OFF), Virtual Reality Modeling Language (VRML), and Drawing Interchange Format (DXF)) to 3D binary uniform voxel volumes. Conversion works best for closed orientable meshes. The output volumes are available in several formats, such as raw binary, VTK, Heritable Image Processing Format (HIPS), and Mimicking Intelligent Read Assembly (MIRA). Apart from voxelization, the toolset also provides a tool for extracting curve skeletons from binary volumes (*thinvox*) and a tool for converting between a number of popular mesh formats (*meshconv*). While the mesh conversion features are less powerful than those provided in related software such as MeshLab, the voxelizer *binvox* offers a very easy to use, robust, and efficiently implemented, way to convert a wide range of 3D meshes to binary volumes up to 1024^3 voxels.
Utilization:	*binvox* is offered as a set of command-line, UNIX-style, utilities that read, process, and write mesh and voxel files. Command-line options are simple and easy to learn and use. As such, these tools can be easily integrated in third-party visualization applications or pipelines. The source code of the toolkit is quite compact, platform independent, and easy to read. This allows one to easily add extra input or output formats and/or integrate it in more complex applications, if needed.

OpenVDB

Type:	Class library
Availability:	Open source (written in C++)
Address:	http://www.openvdb.org/
Description:	Open Volumes with Dynamic Topology, or OpenVDB, is a class library that supports the efficient representation and manipulation of very large voxel volumes. Designed in mind for handling very high resolution sparse volumes, OpenVDB provides a set of sophisticated data storage, indexing, and manipulation operations that allow the processing of 3D voxel volumes of thousands of voxels cubed or more. Operations are provided for creating volumes from a variety of mesh and point cloud formats, reading voxel volumes from third-party file formats, and processing volumes via level set operations, computational solid geometry (CSG) operations, mathematical morphology, and surface advection and tracking. Apart from these, operations are also provided for procedurally compositing volumes and computing various differential quantities on volumes (divergence, Laplacian, curl, and distance transforms). Results

can be exported to both voxel, point cloud, and mesh representations. Support is also included for processing time-dependent volumes, which allows coding various physical simulations that require volumetric domain representations.

Utilization: The main operation mode of OpenVDB is similar to VTK (described above): Users write the intended volume processing scenario in as a C++ program that calls the required functionality provided by OpenVDB classes. Results can be visualized by a number of viewer components provided in OpenVDB itself, or exported to various point cloud file formats. Similar to VTK, this offers a large freedom in building specialized scenarios, but also requires a non-trivial learning curve. While the current focus of OpenVDB is to provide the lower-lever infrastructure required to build end-user applications for volume processing with a focus on volumetric simulations, and less so for interactive volume visualization and exploration, the evolution of OpenVDB will arguably make it easier to use for more general volume processing and volume visualization tasks.

A.4 Grid Processing Software

In this section, we overview several software systems that address the general task of processing *grids*. Under the grid denomination, we include all discrete representations of spatial domains which are formed by vertices connected by various cell types. Following the domain modeling terminology introduced in Chapter 3, we consider here software tools that process discrete representations of 2D curved surfaces (*mesh processing* tools), unorganized point sets, and uniformly sampled 3D volumes (*voxel processing* tools). Given the wide variety of such tools, the focus is here on tools which implement a comprehensive set of typical operations present in grid processing such as resampling, reconstruction, and filtering, rather than on more specialized tools that focus on a narrower set of operations and/or grid types.

MeshLab

Type:	Turnkey system
Availability:	Open source (written in C and C++)
Address:	http://meshlab.sourceforge.net/
Description:	MeshLab is a general-purpose turnkey system for the analysis, processing, and visualization of 3D polygonal meshes. Data can be imported from mesh files in a variety of formats (such as Stanford (PLY), 3D

Studio (3DS), Alias/Wavefront (OBJ), OFF, X3D, and VRML). Both meshes including vertex and cell data, and unorganized point clouds with no connectivity information can be processed. MeshLab includes a large variety of processing operations, including but not limited to mesh cleaning, repairing, simplification, refinement, smoothing and fairing, and computing quality metrics. For point clouds, several algorithms are provided for normals estimation, surface reconstruction, filtering, and registration. MeshLab is continuously extended with recent reseach-grade algorithms via a plug-in mechanism. This makes MeshLab one of the best starting points for applying and/or comparing recent mesh processing algorithms. However, given the rapid pace of development of such algorithms, not all algorithms included in MeshLab have fully optimized or entirely robust implementations. Also, a certain amount of literature study and training is needed to understand the various parameters of the included algorithms.

Utilization: MeshLab can be used much like a traditional image editor. After loading mesh data from files, users can apply any of the provided algorithms in immediate mode, examine the results in a built-in viewer, and repeat the process if desired. MeshLab does not offer the concept of a computational pipeline, such as present in visualization tools such as MayaVi or ParaView. However, this operation mode fits well the highly interactive nature of many mesh processing scenarios, where the user wants to carefully examine the results of each processing step before deciding how and where (on the mesh) to apply the next step, and which this step should be. The final results can be saved in a variety of mesh file formats, compatible with the largest majority of mesh processing or data visualization software tools.

PCL

Type: Class library

Availability: Open source (written in C++)

Address: http://pointclouds.org

Description: PCL (the Point Cloud Library) is a class library dedicated to the acquisition, processing, and visualization of point cloud datasets. Its core focus is on supporting point cloud operations related to typical computer vision use-cases, such as the analysis of, and information extraction from, point clouds acquired with 3D scanning devices such as laser scanners or range cameras. However, PCL components can be also very useful for a variety of operations on (large) point clouds in the context of data visualization, such as point cloud cleaning and filtering, normal estimation, spatial search, registration, segmentation, surface reconstruction, and visualization. Similarly to MeshLab (described above), PCL contains a large set of recent research-grade algorithms, which makes it a valuable resource for the researcher or practitioner interested in testing and/or comparing such algorithms. PCL is designed with scalability in mind, and most of its components can efficiently process point clouds of millions of data points.

Utilization: PCL is a class library, which implies that its users need to program their applications using the provided APIs. Although these are very flexible, learning PCL has a steep curve. The extensive use of nontrivial C++ features and design patterns, and its design that relies on fine-grained components, makes it suitable only for the versed C++ programmer, much like, for example, the Boost C++ library. As its documentation uses a relatively more mathematical presentation angle than typical class libraries, PCL developers should be at least familiar with the main computational geometry concepts and terminology.

CGAL

Type: Class library

Availability: Open source (written in C++)

Address: http://www.cgal.org

Description: CGAL (the Computational Geometry Algorithms Library) is a class library that includes a wide set of algorithms dedicated to the processing of point clouds and polygonal and volumetric meshes. In contrast to MeshLab, for example, CGAL focuses on providing lower-level functionality, or building blocks, that can be used in the development of applications that need to process grids. Included components cover virtually all well-known computational geometry algorithms, ranging from simple spatial searches and intersection computations, Delaunay and Voronoi diagram construction in 2D and 3D, alpha shapes, surface reconstruction, up to complex polygon and polyhedral decompositions, mesh refinement, and surface parameterization. The design of the library makes extensive use of advanced C++ features such as templates and traits. This makes it possible to parameterize the provided algorithms in a variety of directions, such as choosing the space to work in, interpolation type, or numerical approximations to use. CGAL comes with high-quality documentation and an extensive example set, and is actively maintained and used by a sizeable community. As such, it is arguably the tool of choice for application developers requiring non-trivial computational geometry functionality.

Utilization: The main operation mode of PCL is similar to VTK (described above): Users write the intended point cloud processing scenario in as a C++

program that calls the required functionality provided by PCL classes. Results can be visualized by a number of viewer components provided in PCL itself, or exported to various point cloud file formats. Similar to VTK, this offers a large freedom in customizing specialized scenarios, but also requires a non-trivial learning curve.

A.5 Information Visualization Software

Compared to scientific visualization systems, information visualization systems come in a larger variety. There are fewer "generic" systems in this category that can be compared to frameworks such as AVS/Express, SCIRun, or IRIS Explorer. One reason is arguably the higher diversity of the application domains, data types, and end user groups for information visualization systems. Consequently, the selection of information-visualization systems presented next has even fewer pretensions to be exhaustive than our selection of scientific-visualization systems. The considered domains for this selection are graphs and trees, multivariate data, and table data.

The Infovis Toolkit (IVTK)

Type:	Class library/application framework
Availability:	Open source (written in Java)
Address:	http://ivtk.sourceforge.net/
Description:	IVTK is a general-purpose toolkit for developing information-visualization end-user applications and components. IVTK comes as a set of Java class libraries implementing a number of core infovis methods, such as scatter plots, time series, parallel coordinates, matrix plots, and several types of graph and tree layouts.
Utilization:	Developing applications with IVTK and VTK is quite similar. Both are class libraries, so building an application requires programmatically combining instances of the necessary datasets and visualization algorithms. One of the features of IVTK is that it uses a generic dataset model. All datasets (including relational ones) are represented as tables. IVTK provides efficient representations for these tables both in terms of memory and access time. However, just as for VTK, constructing a full-fledged end-user application with IVTK requires a fair amount of work and understanding of the toolkit design. Moreover, compared to VTK, IVTK is relatively newer and less developed toolkit, which provides only a small number of basic versions of the many infovis algorithms that exist for the supported data types (e.g., tree and graph layouts).

Prefuse

Type:	Class library/application framework
Availability:	Open source (written in Java)
Address:	http://prefuse.org/
Description:	Prefuse is a toolkit for constructing information-visualization applications, and is quite similar to IVTK. The toolkit comes as a set of Java class libraries that provides support for representing the main types of datasets used in information visualization, such as trees, graphs, and tables. Together with these, a number of fundamental algorithms for constructing infovis applications are provided, such as graph and tree layouts, glyphs, dynamic queries, brushing, search, database connectivity, and animation.
Utilization:	Prefuse is both a class library and an application framework. Functionality and data representation are provided in terms of classes. Programming interaction, correlation between multiple views, and application execution is provided by means of framework services. In this respect, prefuse is similar to the VTK and IVTK toolkits. However, the architectures and internals of the two toolkits are quite different. A VTK application is structured like a dataflow pipeline. In prefuse, the accent is laid more on connecting data and processing items via actions and events. All in all, prefuse is a good start to learn experimenting with information-visualization concepts and algorithms via prototyping. However, the toolkit does not yet have a wide palette of implemented algorithms, which is similar to IVTK. Also, the scalability and efficiency of the implemented algorithms cannot yet cope with truly large datasets.

GraphViz

Type:	Library and turnkey system
Availability:	Open source (written in C)
Address:	http://www.graphviz.org/
Description:	GraphViz is a high-quality library for computing and displaying graph layouts. GraphViz implements several popular graph-layout algorithms such as rooted and radial trees, hierarchical directed acyclic graph layouts, and force-directed layouts. In addition to layout, GraphViz offers advanced control of the mapping and rendering of graph nodes and edges, including annotations, spline edges, and nested graphs. An extensive set of options allows one to specify the finest details of the layout and mapping. Its robustness, scalability, simplicity of use, and availability have made GraphViz one of the best-known toolkits for laying

	out graphs and quickly producing quality graph visualizations. Several graph visualizations created with the GraphViz software are shown in Section 11.4.2.
Utilization:	GraphViz is structured as a set of separate executables. These read and write graph specification files in various formats. These executables can be easily used as turnkey systems to load, lay out, and draw graphs. In addition to these, GraphViz also provides an API that allows more flexible access to the layout functionalities. This allows one to use GraphViz as a layout library on behalf of other applications.

Tulip

Type:	Library and turnkey system
Availability:	Open source (written in Java)
Address:	http://www.tulip-software.org/
Description:	Tulip is a framework for the manipulation and visualization of large graphs. At the core of the Tulip system is an efficient data representation that allows manipulation of graphs with more than one million elements. The Tulip framework contains a core library and an end-user visualization tool. The library provides graph data representation and so-called algorithms. The algorithms include several layout engines (rooted, radial and bubble trees, treemaps, and force-directed) and rendering engines that allow one to parameterize the node and edge glyphs by graph data attributes. Apart from these, several graph data manipulation algorithms are provided, such as editing, clustering, decomposition, and computing statistics on graphs. Several tree visualizations created with the Tulip system are shown in Section 11.4.1.
Utilization:	Tulip can be used either as a C++ class library or as a turnkey system. In the first case, developers build their application on top of the core Tulip graph data and algorithm classes. In the second case, end users can use the Tulip visualization front-end to interactively import, navigate, edit, lay out, and render graphs in a variety of ways. The functionality of the Tulip front-end, although not covering all the functions of the core library, is rich and customizable enough to allow one to use this application as a full-fledged viewer for complex graphs in real applications. Similar to ParaView and MayaVi, the Tulip front-end can be customized via a plug-in mechanism to load additional functionality developed on top of the core libraries.

Gephi

Type:	Library and turnkey system
Availability:	Open source (written in C++)

Address: http://gephi.org/

Description: Gephi is a framework for the visual analysis of medium to large graphs. In terms of features and utilization mode, Gephi is very similar to Tulip. However, Gephi targets a slightly different user group, and poses the focus more on ease of learn and use than on computational scalability, fine-grained APIs, and algorithm customizability. As such, Gephi offers more plug-ins for importing both static and dynamic graphs from a variety of file formats and live data sources and widgets for interactive graph exploration. However, Tulip offers more research-grade graph layout and analysis algorithms. Also, Tulip is scalable to graphs larger than the ones that Gephi can handle at interactive frame rates.

Utilization: Gephi can be used either as a set of Java class library or as a turnkey system. In the first case, developers build their application on top of the core Gephi APIs (graph, layout, attributes, statistics, import, export, tools, filters, and generators). In the second case, end users can use the Gephi visual front-end to interactively import, navigate, edit, lay out, and render graphs in a variety of ways. The Gephi front-end is very similar (albeit easier to learn but slightly less flexible) than its counterpart in Tulip. The front-end can be directly used to generate a wide palette of graph and network visualization and visual analytics applications. Similar to ParaView, MayaVi, and Tulip, Gephi can be customized via a plug-in mechanism to load additional functionality developed on top of its core APIs.

ManyEyes

Type: Web-based front-end

Availability: Available online as a web application

Address: http://www-958.ibm.com/

Description: ManyEyes is a web front-end for a set of information visualization techniques for interactive exploration of moderately-sized information visualization datasets [Viegas et al. 07]. Provided visualization metaphors include treemaps, node-link graph and tree layouts, bar and line charts, scatter plots, timelines, tag clouds, and data-annotated geographical maps. Each visualization offers a few customization options, such as parameterizing the size, color, annotation, and shape of elements in a node-link layout or treemap; or specifying the columns of a table used to create a 2D scatter plot or bar chart. Customization options can be either explicitly specified by the user, or linked to reflect the value of a data attribute. The created visualizations are displayed online, and can be explored by means of a standard web browser. Interaction features, apart from configuring the visualization parameters, cover interactive zooming, panning, and brushing to reveal data values. The provided visualizations are kept on purpose simpler than the equivalent ones offered by toolkits such as Tulip or Prefuse. However, their built-in default values and presets make them suitable for visualizing a wide range of datasets. Also, the data model is kept very simple: All datasets are basically text documents or two-dimensional data tables. While this offers arguably less freedom to model complex relational datasets, it allows for a very short, easy-to-learn, and error-tolerant path from generating the datasets to creating the actual visualizations.

Utilization: In contrast to most other toolkits, that run as local applications on the user's machine, ManyEyes offers a web-based model: Users format their datasets in a simple, typically text-based tabular model, and upload the resulting data file to the ManyEyes site. Next, visualizations can be created online from the uploaded dataset, both by the user who uploaded data, but also by other users. This model allows for an easy sharing of datasets, constructed visualizations, and insights generated from these visualizations—hence the application's name. The main advantage of this model is the ease by which any user can create a (simple) visualization from tabular data, with zero software installation requirements, and with all the software development and maintenance effort located at the site's provider. Disadvantages involve the need to format the data in the template demanded by ManyEyes; having to share potentially confidential data; and the dependence of a third-party service "in the cloud."

Treemap

Type: Turnkey system

Availability: Open source for nonprofit uses (written in Java)

Address: http://www.cs.umd.edu/hcil/treemap/

Description: Treemap is a customizable turnkey system for the visualization of large multivariate datasets using the treemap layout. Treemap implements several layout algorithms (slice and dice, squarified, and strip) and allows one to parameterize several elements of the mapping process, such as size, color, borders, and labels of the treemap nodes by the data attributes of the underlying tree. Several interactive navigation and filtering mechanisms support a wide range of structure and attribute-based user queries. Treemap also allows one to construct tree hierarchies from data dynamically using a mechanism called flexible hierarchies. Given a set of multivariate data points, trees can be built level-by-level by successively grouping the points by different user-defined criteria on the data attributes.

Utilization:	Treemap comes as a turnkey system that can be customized by means of its user interface. Treemap accepts many data formats as input. Also, Treemap can be configured to monitor "live" data that changes dynamically in time. Its many options can be saved as presets, called feature sets, which allows relatively easy customization without the need for programming. All in all, Treemap is quite easy to use as a customizable turnkey system, but an important limitation is that it cannot be used as a library via an API, e.g., for developing third-party applications.

XmdvTool

Type:	Turnkey system
Availability:	Open source (written in C/C++)
Address:	http://davis.wpi.edu/xmdv/
Description:	XmdvTool is a general-purpose visualization tool for the interactive exploration of multivariate datasets. As such, XmdvTool implements several visualization methods: scatter plots, star glyphs, parallel coordinates, and dimensional stacking. These visualization methods come in a "flat" and a hierarchical variant. The flat variant visualizes all data points separately. The hierarchical variant first groups the data points in a tree, based on some similarity metric defined on the data attributes. Next, tree nodes, which represent data clusters, are visualized using color and shading to map different cluster attributes. XmdvTool is implemented in C++ using OpenGL for the graphics and Tcl/Tk for the user interface functionality.
Utilization:	XmdvTool comes as a turnkey system that can be directly used to visualize multivariate data coming in a number of different formats. The user interface is relatively easy to learn. A strong feature of XmdvTool is the provision of many interaction mechanisms that allow several types of brushing in screen, data, and structure spaces; zooming and panning; display distortion techniques; and masking and reordering of the axes (dimensions). All these mechanisms make XmdvTool a versatile tool that can be used relatively easily to get a first look into a given multivariate dataset. However, just as Treemap, the functionality of XmdvTool is not available as an API or library, which makes its applicability limited in some contexts.