

Exploratory Study of Scientific Visualization Techniques for Program Visualization

Brian J. d'Auriol, Claudia V. Casas, Pramod Kumar Chikkappaiah,
L. Susan Draper, Ammar J. Esper, Jorge López, Rajesh Molakaseema,
Seetharami R. Seelam, René Saenz, Qian Wen, Zhengjing Yang

Department of Computer Science, The University of Texas at El Paso, El Paso, TX
79968, USA

Abstract. This paper presents a unique point-of-view for program visualization, namely, the use of scientific visualization techniques for program visualization. This paper is exploratory in nature. Its primary contribution is to re-examine program visualization from a scientific visualization point-of-view. This paper reveals that specific visualization techniques such as animation, isolines, program slicing, dimensional reduction, glyphs and color maps may be considered for program visualization. In addition, some features of AVS/Express that may be used for program visualization are discussed. Lastly, comments regarding emotional color spaces are made.

1 Introduction

Program Visualization has been defined as the use of various graphical techniques to enhance the human understanding of computer programs [1]. In order to achieve a good visualization of a program, there are three basic visualization stages that need to be taken into account: extraction or data collection, abstraction or analysis, and presentation or display of the result of the analysis [2]. Often, the goals of program visualization reflect the needs of program understanding, design, debugging, maintaining, testing, or code re-use.

Similar to the field of program visualization is scientific visualization. Scientific visualization provides for visual illustration of data and related properties that are often obtained from scientific (or other) disciplines. Scientific visualization helps users to gain a better understanding of complex results. In the past, scientific visualization enjoys more mature techniques, procedures and tools than commonly available in program visualization. Two other differences between the fields include the purpose for the visualizations and dimensional or spatial structuring of the data to be visualized. Whereas scientific visualization is often used to understand the nature of a phenomena, system or application area, program visualization often seeks to provide clarification of a process or procedure so that modifications to the process can be made to generate a 'better' process. In typical scientific visualizations, the data exist in some identifiable structure or coordinate space, e.g., temperature data in a volume. With respect to programs,

there is often a lack of any identifiable structure, that is, the nature of the data to be visualized is not static so much as it is transformable. Program visualization is often about understanding how the data associated with a program is transformed during the execution of the program.

This paper is exploratory in nature. The focus is on program visualization from a scientific visualization context. Its primary contribution is to re-examine program visualization from a different point-of-view; to explore how structure may be added to data associated with a program; and to explore how a well known scientific visualization tool, AVS/Express, might be used in program visualization.

This paper is structured as follows. Section 2 reviews program visualization elements while related work in program visualization is described in Section 3. Section 4 presents a proposed approach to accomplish program visualization from a scientific visualization point-of-view. Section 5 presents a summary of AVS/Express. Section 6 presents various techniques, both scientific and program, that are exploratory in nature but considered potentially useful for program visualizations. An exploratory application example is presented in Section 7. Conclusions are given in Section 8.

2 Program and AVS/Express Visualization Elements

In order to visualize a program, characteristics of the program need to be identified. Characteristics can be categorized as either static or dynamic. Static data that can be visualized in a program involve abstractions such as the following: data structures and attributes, procedures, code metrics and types of operations. Execution statistics are part of the dynamic visualization aspects of a program. Execution statistics involve issues such as: the amount of computations performed, amount of storage space consumed, program control flow and data flow. Debuggers, tracers and other profiling tools are examples of dynamic visualization enabling tools. Animation is frequently used in program visualizations since it is often of interest to study a program's behavior during its execution.

Some AVS/Express features that are under consideration for use in program visualization are now briefly described [3,4]; later sections may provide additional detail. In general, unstructured point data can be represented by coordinate values, multivariate data can be represented by glyphs and line data can be represented by points that describe lines. Colored streamlines with animated markers indicate flow direction. Plotting of surface grids allow for meshes. Arbitrary cut-away 'cuts' provides for a sectioned model. Line graphing facilities are available. Isosurfaces, 3D surfaces with coloring, velocity vectors and contour plots can be used. Discrete data can be interpolated between the points so as to provide more continuous color visualization of the data. Additional techniques include exterior and edge finding; contours, isolines, isosurfaces and isovolumes; slices and cross-sections; colors; 3D scatter data and solid contours in 3D; and city scapes, ribbon plots, and surface plots.

The primary data structure in AVS/Express are fields. Fields combine a structure with data where a structure may be one of: uniform mesh, rectilinear mesh, structured mesh, or unstructured mesh. However, it is not apparent how field structure information can be meaningfully associated with program related data. This subject is addressed in subsequent sections.

3 Related Work

Program visualization is important in the area of parallel and distributed computing. Here, visualization assists in the understanding of such aspects as concurrency, scaling, communication, synchronization and shared variables, see for example [2]. Program visualization is also incorporated into software fault detection [5]. When a failure occurs in a software testing process, it is necessary for fault detection to inspect a large number of software processes from various viewpoints. Program visualization is also used in education, see for example [6].

BALSA is an algorithm animation system that was first developed for educational purposes at Brown University [7]. Although BALSA offers several features that gives it a great value in education, its use is limited by the necessity that BALSA uses a code annotation method. In such a method, the programmer indicates ‘interesting events’ as annotations in the program code in order to mark areas or information that will be used in the visualization. This requires the visualizer to know the program code in order to produce a good visualization of it. The benefit of such a system is to a third party.

A good introduction to the area and much additional related work can be found in [8].

4 A Program Visualization Approach

Given the three stage visualization process of data collection, abstraction and presentation, the proposed approach is: a) identifying relevant program-related information, b) abstracting that information into a form suitable for visualization and c) incorporating specific visualization components for the presentation. Unique in this work is the adoption of scientific visualization techniques and framework for the third stage, namely, the use of a scientific visualization tool for the presentation. In this work, AVS/Express is used in this context.

However, since the data suitable for a scientific visualization has structure, specifically, the data exists in a coordinate space, a requirement is imposed on either the first or second stages to incorporate such a coordinate structure on the program-related information. For example, assume a linear scale and each program statement mapped to equidistant points on the line, then the information ‘program statement’ has been given a (simple) structure. The abstraction stage could augment the information in this example, by say, adding color to distinguish between several types of statements. Such an example (see Section 7), though simple, is suitable for presentation by AVS/Express.

Two key questions are apparent: ‘In which stage ought the structure be incorporated?’ and ‘What would constitute a useful structure?’ The answers to these questions lie outside the focus of this paper, and indeed, are motivations for work-in-progress. None-the-less, preliminary comments can be made regarding the first question; furthermore, this paper is devoted to explorations that address these questions.

Two possibilities exist for the addition of the structure to the data: firstly, in the first stage, the information can be extracted according to a pre-defined structure format; or secondly, in the second stage, a suitable structure can be added to the data. Two examples follow. For the first case, let a 2-D table be identified as relevant, then a coordinate (x,y) can identify an element in the table. For the second case, a control flow graph can be identified based on a standard program analysis where each node in the graph can be identified by an (x,y) coordinate.

Once a coordinate framework has been established, one may imagine the visualization process whereby a) a user creates geometric objects, and b) data is mapped to some geometric features or attributes of these objects.

5 AVS/Express

AVS/Express is a visualization tool developed by Advanced Visual Systems for visualizing scientific data. It uses object-oriented technology, provides for component software development, allows end-user programming and is a visual development tool. The software provides for visualization of complex data and allows applications to incorporate interactive visualization and graphics objects [3]. AVS/Express features a Data Viewer interface, that is, a “point-and-click” interface [4]. This is used for viewing text, graphs, images, geometries and volumes or manipulating display elements such as cameras and light sources. Users have control over the rendering of and interaction with objects in the Data Viewer.

The three programming interfaces in AVS/Express are the Network Editor, the V Command Processor and the Application Programming Interface. As the Network Editor is the principal interface, it is detailed further below.

The Network Editor [3] is a visual programming environment. In the Network Editor, rectangular boxes represent each object of an application. Each object has predefined input and output ports that provide interconnections between objects. Different port attributes, e.g. datatypes, are represented by different colors on the sides of each box. Macros are easily created and stored in libraries for re-use. Connections are made, broken, or re-arranged by “mouse drag and drop” operations. The Network Editor allows the user to modify applications ‘on-the-fly’, with no need to stop, recompile and start again. In the Network Editor objects display their hierarchy and relations in a flow graph-type manner where users may connect objects visually and graphically.

AVS/Express tool kits provide predefined application program components that in addition, enable users to create their own components or incorporate other developed components. Some of the available tool kits include: User Inter-

face Kit, Data Visualization Kit, Image Processing Kit, Graphics Display Kit, Database Kit, AVS5 Compatibility Kit and Annotation and Graphing Kit [3]. Brief descriptions of some of these kits are given below [3, 9]. The User Interface Kit provides the ability to build platform independent or customizable graphical user interfaces. The Image Processing Kit is an extensive library with over 40 image processing functions for analyzing and manipulating images. The Database kit allows for multidimensional visualization applications linked to commercial relational database management systems.

6 Visualization Techniques

6.1 Animation

Animation refers to a process of dynamically generating a series of frames of a scene in which each frame is an alteration of the previous frame [10]. Animation is often used in two different ways in science and visualization: firstly, it is to visualize data to assist in understanding some phenomenon, and secondly, is used to convey information for teaching, recording, etc. In both contexts, animation can be useful for programmers. For example, it can be used to directly support algorithm or data structures teaching. In addition, it can be also used in the analysis of programs for debugging or algorithm enhancement purposes.

Algorithm animation is the process of abstracting the data, operations, and semantics of computer programs, and then creating animated graphical views of those abstractions. A number of program visualization systems such as XTANGO and POLKA [8] incorporated animation. A system called Eliot is an interactive animation environment. Eliot provides a library of visual data types which are ordinary data types with a set of pre-defined visualizations. The user can select one visualization for each data object to be animated. Eliot constructs an animation where the objects as well as their operations are animated based on these selections. Eliot can be used in algorithm design, visual debugging and learning programming [11]. See [12, 13] for further examples. In addition, there are a several web-based algorithm animation systems, see for example Animator [14].

Animation in AVS/Express [15, 16] includes the following four methods. The `Loop` module which outputs a sequence of integers or floats to drive ‘down stream’ modules. The loop controls allow the user to set starting value, ending value and other necessary control information. Animations with `read_Field` can be accomplished by using the temporal data option. The Key frame Animator interacts with modules mainly through the GUI interface. The particle advector animates massless particles along a vector field.

6.2 Isolines

An isoline is a curve that has the property that every point on it has the same value of the *isoline function*. Isolines are a well known visualization component in scientific visualization. Isolines can be used to assist in the visualization of

control flow, static data structure, control flow, etc. For example, an isoline function defined as the (byte) size of every data structures would provide visual information about memory usage (the assumption is made in this and subsequent isoline examples that size exists in a pre-defined coordinate space). In visualizing data flow, isolines could also be used to indicate the data elements that have the same value during the execution of the program; the changes to isolines could provide information regarding how the data values change; the pattern of such a change could indicate program behaviors. Another example of incorporating isolines in a program visualization is the nesting level of program statements; such would indicate areas of nested code.

6.3 Program Slicing

Program Slicing has been used to analyze, debug, and understand computer programs [17]. Program slicing is a “decomposition technique that extracts from program statements relevant to a particular computation” [17]. A typical computer program consists of several smaller sub-programs. Program slicing is aimed at identifying these sub-programs and analyzing dependencies between them. There is much research reported in program slicing, see for example [17]. Program visualization may benefit from the ideas and methods in slicing by visualizing slices of a program then combining these visualizations to produce a whole program visualization. The first two stages of program visualization include data collection and data preprocessing where statements are analyzed to extract data for visualization. Interestingly, part of this work is completed during program slicing, hence, overhead due to program slicing may be reduced by the necessary requirements of the proposed program visualization method.

6.4 Dimension Reduction

For higher dimensional data (i.e., the coordinate space is greater than three dimensions), it is often necessary to reduce the ‘higher’ dimensions to three or two for visual rendering purposes. Reduction of dimension can be done by two methods: focusing and linking [18]. Focusing involves selecting subsets of the data; reduction of dimension by typically a projection or in some cases, by some general manipulation of the layout of information on the screen. Whereas focusing conveys only partial information about the data (i.e., the subset(s)), linking provides for contextual referencing of the data with respect to other data subsets. Essentially, this is accomplished by sequencing several visualizations over time or by showing the visualizations in parallel simultaneously. Specific techniques include dimensional stacking and hierarchical axis.

6.5 Glyphs in AVS/Express as a Means of Representing Higher Dimensional Discrete Data

A glyph is a single graphical object (e.g., sphere, diamond, square) that represents a multivariate data object/value at a single location in space. Multiple

data attributes may be mapped to appearance attributes of the glyph (e.g. size, color, texture). A glyph can be used to represent a higher dimensional data set. See [19] for details regarding glyph representation.

In AVS/Express, the module GLYPH located in the main library, mappers section, can be used to represent higher dimensional data. It has three input ports one of which is called the `in_field` port that takes any mesh plus node data, another is the `in_glyph` port which takes as input an unstructured mesh that forms the geometric object used as a glyph. The parameters of the glyph module provide for size, color orientation of the glyph. AVS/Express is capable of representing data sets of multi-discrete data of up to 10 dimensions (i.e, `position=3`; `color=3`; `shape/size=1`, and `orientation=3` — refer to [19]).

Using glyphs in AVS/Express may be accomplished as follows. The input field `in_field` takes as input one or many field type data structures that are output from other modules. Specific AVS/Express information include: a) Glyph Component: will determine the scale/size of the glyph; b) Map Component: selection of the data component from the input field(s) to color the glyph; and c) Mode: determination of how the module will portray the data values. By choosing a vector, a vector type object indicating the orientation of the glyph may be obtained.

6.6 Color Maps

Color sequencing in maps is important for proper understanding of the data attributes. Pseudosequencing is a technique of representing continuously varying map values using a sequence of colors. For example, geographers use a well-defined color sequence to display height above sea level: lowlands are green (which illustrates images of vegetation); browns represent mountains, etc. Such a sequence follows a (logical) perceptual sequence often made understandable with training.

In pseudosequencing a map, there are generally two important issues. The first is perceiving the shape of features. The second is the classification of values on the basis of color. If the color sequencing is poor, the understanding of the map tends to be more difficult. One important factor is that colors used in maps follow a perceptual (or human understanding) sequence. Several alternatives include the luminance sequence for representing large numbers of details; and a chromatic sequence. when representing little detail. In general, a perceptual ordered sequence will result from a series of colors that monotonically increases or decreases with respect to one or more of the color opponent channels. Also, in general, the larger the area that is color coded, the more easily colors can be distinguished. When large areas of color coding are used with map regions, the colors should be of low saturation and differ only slightly. (See [19])

6.7 Emotional Space Color Order

The feelings that humans experience such as happiness, sadness, loving, etc. are emotional behaviors. Emotional behaviors are powerful, in part, because of its

impact. An interesting point to note is that color evokes emotion [20]. When humans talk about color for example, color becomes a descriptor of the physical world. Individuals have different interpretation of the different colors, which means that color is also a visual response to physical data; an emotional response to expectation and violation of appearance. It is of interest to consider that a color-based visualization may be used to provide emotional stimuli regarding user-feelings about the program (as for example, the common situation whereby a programmer firstly encounters a poorly written code fragment and has a negative emotional response).

The emotions that are experienced can be modeled as emotional space. One example of this is the relationship between color and sound: the voice of a child may be connected to the color yellow [21]. There are two important issues related to this idea of emotion in color communication. Firstly, it is difficult to reproduce a color message in different media, i.e., to select an alternative stimuli requires some ‘emotion metric’. Secondly, composing a presentation of color is difficult. The message creators sometimes generate the unintended emotional message for lacking of guidance in selecting colors. To translate the relationship into the desired emotional response, a message creator needs to manipulate the stimuli properly.

According to Mendlers theory of emotion [20], the emotion system or the human feeling of emotion has an internal (human) sense-of-order or expectation. Emotion can be ‘increased’ or ‘decreased’ in the processing of the information conveyed by the colors. This information needs to have color relationships that fits some sense-of-order. To define an sense-of-order is insufficient to induce an emotional response. Additionally, resolution is required. Resolution is the effect of perceived color with respect to the sense-of-order.

For useful application to program visualization as presented in this paper, it is of interest to note if there are models that represent human sense-of-order interaction. Several tests have been done [20] regarding the questions like: do color interactions exist? What is the space definition of the color? What needs to be determined along a color dimension? How to measure a violation of the color order? Some reported results indicate that a color change in each dimension can be used as a frame work to set up the color expectation. When an expectation differs from the perceptions, an emotion may be generated.

7 Example Application

Figure 1 presents an example visualization of static, program-related data (this visualization was not done in AVS/Express). The horizontal axis refers to the time of the program’s execution; the vertical axis, to the amount of resources consumed; and the depth axis, to the nesting of statements. Each statement is depicted by a rectangle placed on the horizontal-depth plane (the width of the rectangle is constant since no data has been mapped to this parameter) The colors are mapped as follows: gold – input statement, yellow – output statement, green – computational statement and light-blue – control statement (if gray-

shaded, the classification is ordered by increasing grayness in the order of output, input, control and computation statements). The visualization shown indicates a typical program with the following characteristics: the program starts with an input statement followed by three computational statements at a nesting level of one. The next statement is a control statement at nesting level of two. Two subsequent computational statements execute in a nesting level of three, these statements consume the most resources out of any other statement. the program completes by an output statement, input statement, computational statement and lastly, an output statement.

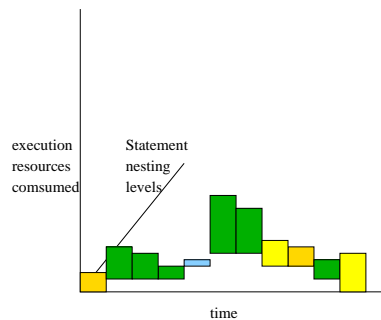


Fig. 1. A program visualization example.

This example illustrates several of the aspects described in this paper. First, program statements have been structured in a 1-D space (the horizontal axis). Second, two additional dimensions represent statement properties. Third, a colorization based on a statement category is used (however, it should be pointed out that the selected color map used in this example is *ad hoc* in nature).

8 Conclusion

This paper presents a unique point-of-view for program visualization, namely, the use of scientific visualization techniques for program visualization. This study confirms that the key aspect in integrating scientific visualization techniques in program visualization is the incorporation of structure, in particular, coordinate based data, into program-related data. This paper explores basic and fundamental issues and techniques from both scientific and program visualization in the pursuit of integrating the techniques. This paper reveals a number of interesting aspects that may be fruitful to consider in the future. Specific visualization techniques are presented and explored for utility in program visualization. This paper also describes certain features of AVS/Express that may be used for program visualization. Lastly, some additional comments regarding emotional color spaces are made. Future work to continue the investigation into scientific and program visualizations is indicated.

References

1. B. A. Price, R. M. Baecker, and I. S. Small, "A Principled Taxonomy of Software Visualization," *Journal of Visual Languages and Computing*, Vol. 4, pp. 211–266, 1993.
2. E. Kraemer and J. Stasko, "Issues in Visualization for the Comprehension of Parallel Programs," *Proc. of the 3rd Workshop on Program Comprehension*, Washington, D.C., Nov. 1994, pp. 116–125, Nov. 1994.
3. Advanced Visual Systems Inc., *Using AVS/Express*, r. 4.0 ed., 1998. Part No. 320-0321-05 Rev. A.
4. Advanced Visual Systems Inc., *Visualization Techniques*, r. 4.0 ed., 1998. Part No. 320-0324-05 Rev. A.
5. H. Amari and M. Okada, "A Three-dimensional Visualization Tool for Software Fault Analysis of a Distributed System," *Proc. of the 1999 IEEE International conference on Systems, Man, and Cybernetics*, Tokyo, Japan, Oct., 12-15, pp. 194–9, Oct. 1999. Vol. 4.
6. P. Smith and G. Webb, "The Efficacy of a Low-level Program Visualization Tool for Teaching Programming Concepts to Novice C Programmers," *Journal of Educational Computing Research*, Vol. 22, No. 2, pp. 187–215, 2000.
7. M. Brown, "Exploring Algorithms Using BALSAs-II," *IEEE Computer*, Vol. 21, No. 5, pp. 14–36, 1988.
8. J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, (eds.), *Software Visualization, Programming as a Multimedia Experience*. The MIT Press, 1998.
9. Advanced Visual Systems Inc., *AVS/Express Toolkits*, r. 4.0 ed., 1998. Part No. 320-0323-05 Rev. A.
10. N. Magnenat-Thalmann and D. Thalmann, *Computer Animation Theory and Practice*. Springer Verlag, 2 ed., 1991. ISBN: 038770051X.
11. S. Lahtinen, E. Sutinen, and J. Tarhio, "Automated Animation of Algorithms with Eliot," *Journal of Visual Languages & Computing*, Vol. 9, pp. 337–49, June 1998.
12. C. D. R. Baecker and A. Marcus, "Software Visualization for Debugging," *CACM*, Vol. 40, pp. 44–54, April 1997.
13. O. Astrachan and S. Rodger, "Animation, Visualization, and Interaction in CS1 assignments," *ACM. Sigcse Bulletin*, March 1998, Vol. 30, pp. 317–21, March 1998.
14. <http://www.cs.hope.edu/~algaanim/Animator/Animator.html>.
15. <http://www.osc.edu/~kenf/Visualization/XPsciviz/sv-schedule.html>.
16. http://www.ncsc.org/training/materials/express_class/XPsciviz/XPia/xpia11-frame.html.
17. D. W. Binkley and K. B. Gallagher, "Program Slicing," *Advances in Computers*, Vol. 43, 1996.
18. R. Wegenkittl, H. Loffelmann, and E. Groller, "Visualizing the Behaviour of Higher Dimensional Dynamical Systems," *Proc. of the 8th IEEE Visualization '97 Conference*, Oct, 19-24, 1997, pp. 119–125, Oct 1997.
19. C. Ware, *Information Visualization Perception for Design*. Morgan Kaufmann Publishers, 2000.
20. B. Burling and W. R. Bender, "Violating Expectations of Color Order," *Proc. SPIE Vol. 2657, Human Vision and Electronic Imaging*, April 1996, B. E. Rogowitz and J. P. Allebach, (eds.), pp. 63–71, April 1996.
21. T. Miyasato, "Generation of Emotion Spaces Based on the Synesthesia phenomenon," *Proc. of the IEEE Third Workshop Multimedia Signal Processing*, 1999, pp. 463–7, 1999. Cat. No. 99TH8451.