Stencil: A Conceptual Model for Representation and Interaction

Joseph Cottam*, Andrew Lumsdaine†
Open Systems Laboratory-Indiana University
Bloomington, IN, USA

Abstract

Existing Information Visualization models provide insufficient support to visualization programmers in creating applications. They either broad and taxonomy based, or narrowly focused on isolated aspects of the visualization problem. Taxonomy based tools are good for categorizing what has been or needs to be done, but provide little help in accomplishing those goals. Narrowly focused models allow particular aspects of the visualization problem to be efficiently solved, but leave heavy burdens for integrating many such narrow tools together to solve the overarching visualization problem. This insufficiency of visualization models is most evident where interaction is concerned: It is often left as an afterthought. In this paper, we describe the Stencil visualization model, an intermediate model that covers many visualization specific issues. We argue that the Stencil model can guide visualization program construction through several stages of common application pipelines; thereby improving the resulting visualization products and reducing significant barriers to visualization adoption.

Keywords—**Declarative Language; Models**

1 Introduction

Information Visualization is often defined as $representation + interaction$, but the interaction is often given only secondary attention [20]. Scientific Visualization has developed techniques to process and visually represent some abstract data sources effectively. However, in Information Visualization, representation is only part of the problem; the interaction component is often either relegated to secondary status or ignored entirely. This is probably due, at least in part, to the fact that the programming models for Information Visualization give no tools to deal with the interaction at a higher conceptual level. In this paper we outline the Stencil Model, a conceptual model that begins to address this and other Information Visualization problems.

*jcottam@cs.indiana.edu
†lums@cs.indiana.edu

The downfall of contemporary visualization models is that they do not include interaction in a conceptually coherent manner, but rather they rely on the facilities of the programming language. This lack of attention is likely driven by the abstraction levels selected by the model designers being either too broad or too narrow. Those that work too broadly try to encompass all of the (necessary) data processing tasks that go into effective visualization, but only lightly deal with interesting aspects of individual steps in that process. They effectively state what must be done, but do not give clues on how to accomplish it. Conceptual models that focus too narrowly tend to only address issues of representation, and often with an eye towards computationally efficiency. We feel that the field of Information Visualization would benefit from a model that operates at an intermediate level, directly addressing issues of representation, refinement and interaction. Clues to the components of such a model can be drawn from the fields of Graphic Design (professionals in visual communication and graphics mediated interaction) and prior Computer Science successes in interaction modeling.

In this paper, we examine the shortcomings of existing models in more detail and we then look at some of the relevant lessons of Graphic Design. After that, we will describe the foundations of the Stencil model, including how it captures representation and interaction concerns. We then describe our early experience with a precursor model and conclude with a discussion of the next steps.

2 Related Works

The related works fall into two general categories. First we examine the current visualization models and discuss their shortcomings. Second, we will look at related fields, highlighting strengths in them that may address the shortcomings of the existing models.

2.1 Existing Models

Existing models for Information Visualization can be divided into two categories: Graphics Libraries and Whole Process. We will look each of these categories separately.

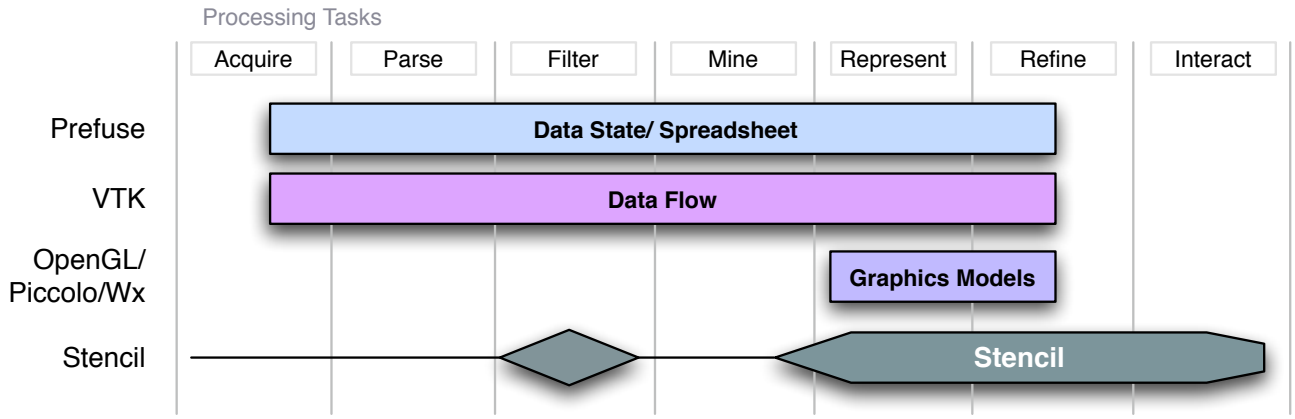

Figure 1: Example visualization software aligned to the Processing pipeline stages. Thought VTK and Prefuse include interaction, it is not derived from their model (and thus not include above). Further, the VTK and Prefuse implementations focus on representation, not refinement. Altering a representation often takes as much effort as the original definition. Stages/tasks and alignments were derived from [8, 10, 4, 16]. In contrast, the Stencil model directly supports refining and interaction while retaining support for representation and filtering. The parse, mine, and acquire stages are not directly supported in the model, but the Stencil system provides links to call software that may. This selective attention and partial coverage aides in simplifying the model. The links to external code allows this simplification without a corresponding loss of expressive power in a Stencil-based application.

An overview of the concerns of these models is given in Figure 1.

Graphics libraries (like Piccolo, Wx, OpenGL, etc.) focus on the act of rendering graphic entities to the screen and the ability of programmers to describe what needs to be rendered. A variety of approaches have been explored, such as matrix transformations [3], scene-graphs [1] and entity tables [9]. Each of these models for handling the display problem comes with strengths and weaknesses particular to their methods. However, these techniques only focus on the *representation* portion of the visualization problem. Having such tools available as part of an Information Visualization solution has allowed attention to be paid to other parts of the visualization process. However, by not including other late-process issues (e.g. interaction and refinement), they are incomplete guides for creating fully functional Information Visualization applications. These models are effective in their limited domain, but require great effort to be integrated with other models in intrinsically related domains.

In contrast to the highly specialized graphics models, several models for the entire Information Visualization process have been developed. Most notable are the data-state and data-flow models. The data state model treats data tables as monolithic entities, logically occupying separate states though they represent the same base data. Transitions between data states operate on the entire table and produce an entirely new table. The ordering of the logical states is from base data to progressively greater analytical abstractions [4, 10]. The data flow model is the dual to the data state model, treating data operators as entities and with individual data points 'flowing' through networks of these operators [16]. The expressiveness of these two models has been shown to be identical [4].

Many other models can be expressed as refinements of these, providing more sub-divisions in the logical groupings of either operators or data states. This includes the recent Processing model [8]. Broadly speaking, models in these families try to capture the process of Information Visualization from start to finish (i.e. data creation to interactive application). These models generally suffer from the opposite problem of the graphics library models; they tend to be too general. They are suitable for taxonomy creation, giving a categorization scheme that handles every step, but do little to guide the actual resolution of Information Visualization problems. Significantly, these models leave all of the details up to ad-hoc programming constructs. In the end, programming libraries tend to focus on individual phases of the described visualization process. Lacking data structure, control flow or even conceptual consistency between tools, a visualization programmer's main task becomes understanding and managing these complex relations in an *ad-hoc* fashion as well. This is especially significant where interaction is concerned because the majority of visualization libraries simply rely on the raw interaction primitives of the implementing languages.

2.2 Directions for Improvement

Many visualization systems are far removed from the practice of Graphic Design [19, 15, 11]. This is unfortunate because graphic designers are professional visual communicators , and their practices likely hold many clues to effective visual communication [11].

Traditional software library design has tried to bring programming to the practice of visual communication. However, the opposite problem, bringing visual communicators to programming techniques, has been undertaken by Fry with the Processing system [8]. Contributions of Processing include a flexible graphics library that facilitates graphic design considerations (such as variable registration points when specifying geometry) and indications that visualization will yield to high-level semantics in some areas (Processing effectively abstracts out class hierarchies and some multi-threadin issues). However, the conceptual model of the majority of the Processing language and supporting development application are identical to those found in Java.

Before a tool can be expertly used, the the system-level effects of that tool need to be well understood. In computer programming, this often means that understanding the hardware behavior is essential to expert programming. In visualization research, the corollary discussions are those of semiology and psychology. The psychological aspects of visualization are thoroughly explored in [18]. The semiology of graphics is dissected in [2]. Automating the consideration of issues from both fields of effect has been attempted, notably by Mackinlay [13]. The need for such understanding and the potential for the automation that Mackinlay approached was also noted by Watson [19]. Watson further notes that effective automation will require a better description of the context that a visualization will be exercised in.

3 The Stencil System

The Stencil system aims to reduce the problems associated with adoption of visualization by moving issues peculiar to visualization in to the foreground while retaining access to the power of existing algorithms. We employ a declarative Domain-Specific Language (DSL), called the Stencil Language, with facilities to access functions written in a general purpose language. The Stencil Language holds closely to the conceptual model we are concurrently developing, and presenting here (e.g. the Stencil model). This concurrent development allows us to test the components of the model from both a conceptual and a technical standpoint. Through the Stencil system we hope to enable rapid visualization prototyping with real-world data, enable synthesis of multiple data sources [17] and decouple visualization descriptions from specific graphics libraries. The details of the language and system architecture are left for future venues, and the remainder of this paper will focus on the Stencil model itself. An overview of the feature distribution of the stencil model can be found in Figure 1.

3.1 The Stencil model

To overcome the interaction shortcomings of existing models and make the practice of graphic design more directly accessible, we propose a new conceptual model for visualization. This model, called the Stencil model because it is being developed as part of our Stencil visualization system, has three major (distinct) features. First, it exposes a revised view of the problem of visualization n terms of the common graphic design tool layering. Second, the data model adapts directly to common user input methods. Finally, the model explicitly includes some properties of the display device. The following section describes the features of the Stencil model in more detail and how they help solve the above-mentioned problems.

The following concepts are the starting point for a model to reify in the Stencil language. These core concepts are believed to capture many of the core concepts of interactive visualization. Concept development has been based on the experience of a preliminary declarative visualization system implemented in the Summer of 2007 called 'ThisStar' (discussed further in Section 4).

The major data abstractions for the Stencil model are:

Tuples are the basic data unit. A tuple is a compound data element where the order of the constituent data points is significant. Classic examples are coordinates and database record-set rows. Tuples are thus analogous to the 'information packet' found in flow-based programming [14].

Streams represent ordered groups of tuples. A stream is a forward-only information source, items once seen cannot be recalled unless they are explicitly stored. Streams are the data interface between Stencil and the rest of the world. All information entering or leaving a Stencil-based application is represented as a stream.

Logical graphical elements comprise the items that will eventually be rendered. The following logical elements are anticipated in the model:

Glyphs are visual representations of data (e.g a circle, a dot, text, etc). The Stencil model converts abstract data tuples to glyph tuples for later rendering. Glyphs all share a minimum set of attributes (e.g. ID and position). Additional attributes specialize glyph types and may include coloring, mark styles, or labeling.

Glyph Groups create collections of related glyphs. Glyph groups are glyphs themselves. The ID of a glyph

group is a namespace identifier for the glyphs it contains, this allows glyphs to be uniquely picked by its path from the root through all of the enclosing groups.

Layers form the root of the group hierarchy. Layers are a special type of glyph groups, in that they contain glyphs. However, layers (and only layers) may appear as children of the root of the glyph hierarchy. As implied by the name, layers also form the primary Z-ordering unit.

Transforming tuples from streams is achieved via a **mapping operator**. Data tuples enter a mapping function that produces new tuples on exit. This is analogous to the operators found in data-flow languages. Similarly, mapping operators may be chained to create complex results. The mapping operator provides a means to leverage arbitrary algorithms, as they are only required to take and return tuples. Mapping operators also pull additional duty as a filtering construct, since mapping to an invisible glyph has the same effect as filtering the data. As such, eventually all tuples on a stream will either be applied to glyph attributes or filtered out by the mapping operators.

3.2 Rendering and Interaction

To facilitate the physical display of information on the limited display spaces actually available, the following conceptual additions are under consideration for the Stencil model:

Layers carry additional responsibilities from an interaction point of view. As mentioned, the full name of a glyph is the path through all of its containing groups. The root of this path will always be a layer; as such, layers expose the search functionality. The natural outgrowth of that ability is the selection capability that forms the basis of many interactive operations. Also, the ability to modify glyphs after they have been initially created will rely heavily on this search ability.

Canvas defines the rendering space for the eventual marks. All layers cover the entire canvas (regardless of their coordinate systems) but may have different origins. The canvas is a logical space, the *view* is the actual rendering of a section of the canvas. The canvas sets properties for the entire visualization.

Views are a slice of the canvas as it is rendered onto a device. This is a page from a printer, the bounds of a file or a window on a screen. Views form the basis of the visual interaction and represent the only implicitly non-stream output of a stencil component.

The combination of the proposed data model and the display elements provide the foundation of model-based interaction. Simply put, data streams form the input and display objects form the output; interaction is then composed of mappings between inputs and display object states. Modeling keyboard as a streams is a long used pattern [12]. Using a stream of keystrokes could be thought of a stream of 1-tuples. Modern GUI programs model the keyboard input as a series of keystroke events through callbacks. Even in an event system, keyboard events can be thought of as a stream of event-descriptors instead of discrete events. Event descriptors include information like modifier keys and focused controls, but these can be represented as tuples. Similarly, the mouse events can also be represented as a stream of tuples. Using these event descriptors to control the display characteristics by manipulating view properties allows zooming, panning, etc. Where event descriptors include 'picked' items, brushing/pouring and dynamic highlights also become available. Tying the events associated with traditional GUI elements (like text boxes or radio buttons) to filtering commands concisely expresses the ideas behind dynamic queries. As such, the Stencil conceptual model brings direct access to common interaction techniques in Information Visualization without multiplying conceptual elements beyond those required for representation.

4 Indications of Efficacy

The model described above is derived from the our experience with the experimental ThisStar application [6, 5]. ThisStar was designed to allow rapid visualization development of point-implantations [2] using simple or precomputed layouts. It was intended to fill the gap between full visualization applications and off-line data-anlysis algorithms. The observation that led to the development of ThisStar was that bringing the results of off-line analysis (i.e. from statistical packages or non-visualization domain libraries) into a visualization application is bringing groups of tuples from their own abstract spaces and joining them in the same graphical space. This idea is conceptually simple, but proved complicated in practice. ThisStar allowed point implantation graphics (e.g. scatter plots or star maps) to be created from any layout generator and combined with the meta-data from other analysis. ThisStar permitted simple modifications to incoming data computed (e.g. filtering, point changes to data and some uniform distortions) and a few basic layouts could be directly expressed in its declarative language (e.g. categorical grids) . However, ThisStar was severely limited by both its model and its implementation. Its successes, shortcomings and influence on the Stencil model are described in remainder of this section.

The largest success of ThisStar has been the effectiveness of the declarative language that formed the basis of

the interface. First, this programming style allowed us to rapidly prototype divergent ideas for quick tests using real data. Simple changes to the representation often only required simple changes to the description, this includes changes to the pre-computed layouts. Furthermore, the layering mechanism facilitated incremental construction without risk of interfering with prior-made results. This allowed us to quickly try many ideas with real-world data. The second major advantage of using a declarative language in our ThisStar interpreter was the completeness of the eventual description. This advantage of declarative languages is best understood by contrast to typical interpreters. During interactive interpreter sessions in GUESS or R, the entire session trace must be considered if the results are to be reproduced. Reducing this to just the significant commands is time-consuming and error prone. By contrast, with a declarative language all commands are re-evaluted as a whole. As such, we retained the fast-iteration benefits of using an interpreter, but reapplication is trivial. Everything that is left in the command window at the end of a run contributed to the result. Saving this final state allows results to be recreated at a later date, or the same process may be applied to new data and.

ThisStar was implemented in terms of both the Piccolo and the Prefuse frameworks. These two frameworks having substantially different low-level models. The visualization Prefuse model was very tied to Prefuse data structures and was hard to abstract. The Piccolo model (a scene graph library) is very focused on efficient graphics processing, but does not have the breadth to handle the issues of converting abstract data into graphics. In the end, we developed a third conceptual model to abstract over the low-level details of the individual libraries and present a set of abstractions suitable for declarative visualization specification. This observation has lead directly to the Stencil model development.

The Streams and Tuples data representation in the language of ThisStar is similar to that of the Stencil model. In ThisStar, tuples were represented as name-indexed value groups rather than position index value groups. In practice, these were interchangeable and may be so generally. The data coordination engine of ThisStar was incapable of handling more than one infinite streams, however this limitation was only present in the data coordination engine, not the interpreter of the declarative language. Improving this coordination engine will be essential for the Stencil system, especially for capturing interaction. However, the conceptual simplicity of Streams and Tuples was useful and has been retained.

A major deficiency of the ThisStar model was its poor grouping constructs. The only grouping construct in ThisStar was the layer, which controlled Z-order and little else. Without grouping, many useful elements (like recursive layouts or dynamic gridlines) is difficult. The intermediate Glyph Group is one of the major advances for the Stencil model, its inclusion is a direct result of the ThisStar experiment.

Missing entirely from the ThisStar model were the explicit canvas and view objects. Though ThisStar included some rudimentary user input faculties (i.e. the keyboard and mouse states could be accessed as special streams), the inability to access the state of the visual output made most interaction techniques inaccessible. Concepts as simple as geometric zoom and pan could not be expressed.

In summary, the ThisStar application was a successful prototype that has helped us explore the requirements of a visualization model that includes user input and graphic design concepts. However, the Stencil model goes far beyond ThisStar and will enable many more visualization techniques to be more directly addressed.

5 Future Work

The most important future work is to develop a system based on the revised model described here. The ThisStar system showed that the data abstractions being employed were effective and that better refinement support are possible. However, the ThisStar language and system are not able to capture many useful visualization techniques. In a similar way, the concepts presented here are not likely to be comprehensive (as the Stencil system is a work in progress); constructing a functional system based on the presented ideas will help expose the shortcomings and give insight into how to address them.

In its present state, the Stencil model is isolated from other visualization models. How it relates to both the data flow and data state models would place it more securely in the visualization landscape. Such formal situation will help identify when Stencil should be used instead of one of these other models.

Creating effective support tools for working with either a new version of the Stencil language or a library based on the core principles of the Stencil model is another important area of exploration. Since the Stencil model does hold closer to common Graphic Design metaphors, it may be useful for broadening the class of people who can employ visualization directly. To reduce the barrier to entry even farther, composing Stencil model code through visual means would is approachable through the specification-oriented nature of a declarative language, which Stencil provides.

A final important future work would be the evaluation of applications employing the Stencil model. Such an evaluation requires applications to actually be constructed that employ it. Evaluation dimensions of importance include traditional memory and speed considerations (and their

first level-derivitives of maximum data size for interactive visualization). Another, less traditional dimension, would be time-to-creation of the software itself. Such evaluations are the focus of 'high productivity' software development research [7]. The language-based aspects of the Stencil model and related tools were conceived to help improve the visualization product-development cycle. Evaluating their success in these terms will be instructive in further development of Stencil and similar tools in other domains.

6 Conclusions

We have discussed the fundamental shortcomings of current visualization models to support interaction and declarative visualization. We have outlined a the Stencil model; a visualization model designed to ameliorate these issues. We have discussed the ThisStar system, which employs a precursor to the Stencil model, with the successes and failures found there. We believe that the Stencil system, will enable rapid development of novel visualizations over diverse data. This will expand the presence of interactive Information Visualization and encourage more people employ it in their own processes.

Acknowledgements

This work was supported in part by a grant from the Lilly Endowment.

References

[1] Benjamin B. Bederson, Jesse Grosjean, and John Meyer. Toolkit design for interactive structures and graphics. *IEEE Transactions on Software Engineering*, 30(8):535–546, 2004.

[2] Jean Bertin. *Semiology of Graphics*. Reprinted by University of Wisconsin Press, 1967.

[3] OpenGL Architecture Review Board, Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison-Wesley Professional, 2007.

[4] Ed H. Chi. *A Framework for Visualizing Information (Human-Computer Interaction Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[5] Joseph A. Cottam and Andrew Lumsdaine. ThisStar: Declarative visualization prototype. In *IEEE Symposium on Information Visualization (INFOVIS'07)*, 2007.

[6] Joseph A. Cottam and Andrew Lumsdaine. Tuple Space Mapper: Design, challenges and goals. Technical Report TR648, Indiana University, Bloomington, IN, June 2007.

[7] DARPA. High productivity computer systems. http://www.HighProductivity.org, 2008.

[8] Ben Fry. *Computational Information Design*. PhD thesis, Massachusetts Institute of Technology, 2005.

[9] Jeffrey Heer and Maneesh Agrawala. Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):853–860, September/October 2006.

[10] Jeffrey Heer, Stuart K. Card, and James A. Landay. Prefuse: A toolkit for interactive information visualization. In *Proceeding of the SIGCHI Conference on Human Factors in Computing Systems (CHI'05)*, pages 421–430, New York, NY, USA, 2005. ACM Press.

[11] Robert Horn. *Visual Language*. Macro Vu, Inc, Bainbridge Island, Washington, 1998.

[12] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988.

[13] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141, 1986.

[14] J. Paul Morrison. *Flow Based Programming: A new approach to application development*. Van Nostrand Reinhold, 1994.

[15] Holly Rushmeier, Jason Dykes, John Dill, and Peter Yoon. Revisiting the need for formal education in visualization. *IEEE Compuer Graphics and Applications*, 27(6):12–16, November/December 2007.

[16] Will Schroder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics (3rd Edition)*. Kitware, Inc., USA, 2002.

[17] James J. Thomas and Kristin A. Cook, editors. *Illuminating the Path*. IEEE Press, 2005.

[18] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufman, San Francisco, 2000.

[19] Benjamin Watson. Broadening our collaboration with design. *IEEE Compuer Graphics and Applications*, 26(5):18–21, September/October 2006.

[20] Ji Soo Yi, Youn ah Kang, John Stasko, and Julie Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, 2007.