A Buffer Framework for Supporting Responsive Interaction in Information Visualization Interfaces

Tobias Isenberg André Miede Sheelagh Carpendale
Department of Computer Science
University of Calgary, Canada
{*isenberg* | *amiede* | *sheelagh*}*@cpsc.ucalgary.ca*

Abstract

We present a framework that we are developing to better solve several critical issues that arise when interactive systems are extended to large displays. These issues include slow reaction times, difficulties with high numbers of concurrent interactions or user inputs, and problems that occur when combining several aspects of visualizations. In part, these issues arise from a number of complexities that are present in current approaches. This makes it important to tackle this problem directly rather simply waiting until the computing power has increased sufficiently and calls for a fundamentally new approach to computer interface foundations. Our framework combines ideas from information visualization, large displays, collaborative work, and non-photorealistic rendering (NPR). Specifically, we are employing four concepts/techniques: layered buffers, local coherence, emergent complexity, and force fields.

Keywords: Large displays, responsive interaction, human-computer interaction, buffer framework.

1. Introduction and Motivation

Much of present-day intellectual work is done in teams and requires that information be presented in ways that support shared analysis and decision-making. Computer graphics, information visualization, and human-computer interaction (HCI) research fields are converging to solve this important problem. Several key advances have been made in solving individual parts of the larger problem:

- visualizations concentrate on supporting the analysis and interpretation of data needed for decision processes by presenting it in accessible formats [11, 12],
- large screens are being developed in different formats (walls, tables, etc.) accepting input from multiple sources to support team work and collaborative processes [4, 18],
- visual setups designed to better support collaboration by providing awareness information about each other's work in progress and to attempt to generally reduce cognitive load [9, 16], and
- improvements in NPR graphic methods that address clarification of chosen aspects of depicted items rather than aiming at fully realistic presentation [19].

However, integrating these independent advances to fully satisfy the needs of work groups is difficult due in part to the ways in which present-day user interfaces generate visualizations and provide interaction with them.

In addition, due to complexities in the system design of current visualizations, user interfaces are severely limited in terms of further extension. Modern computer interfaces typically use two types of entities to generate visualizations, to facilitate collaboration, and to support user-interaction: the actual *visualization objects* that carry the information to be displayed and the *interface components* that guide the behavior of the visualization objects, which are in turn affected by user interactions. However, this approach has lead to the following four major limitations:

- limit in the number of visualization objects because they have to be steered and maintained by a complex and application-specific interface components,
- complexity of interaction between visualization objects and interface components due to the increased need for maintenance as the size and detail of interface components increases,
- complexity of interaction between several interface components since this leads to an exponential computation effort when combining them to jointly control visualization objects, and
- limit in the number of simultaneous user-interactions with either visualization objects or interface components due to the specific architectures of the interaction between objects and interface components.

In addition to these system-internal complexities, the development and extension of interfaces for large displays consequently also becomes an increasingly complex endeavor.

Thus, today's interactive visualization environments, when extended to large displays, are not successful in providing timely feedback to multiple users who are concurrently interacting with a high number of objects. We address these problems by providing a framework that reduces the mentioned complexities.

In the following, we first discuss some related work in Section 2. Then, we present an overview of our overall concept in Section 3 before giving more details about our system and its software architecture in Section 4. Then, Section 5 discusses potential application domains, our implementation, and reports some initial results. We finish with a conclusion and some future work in Section 6.

2. Related Work

The work presented in this paper ties together ideas from different areas in computer science. We briefly present the underlying concepts in this section.

Our system relies first and foremost on the concept of buffers that store data in form of two-dimensional images, a technique used very frequently in computer graphics. There, other data about the rendering process is stored in addition to the actual image buffer to be used for additional computations or as additional results. The first of these buffers—the z-buffer—was introduced by CATMULL [3] to store depth information to facilitate the hidden surface removal that is necessary for rendering. The concept was later extended by SAITO and TAKAHASHI to store any additional information in form of G-buffers which they used to extract silhouettes and feature lines [13]. Since then, many new uses for G-buffers have been explored.

We also build on ideas from swarm intelligence. There, complex behavior emerges from a swarm of individual members each with only limited intelligence and without a concept of the swarm as a whole [2]. This idea has also previously been used in connection with image buffers to facilitate the simulation of non-photorealistic rendering styles using RENDERBOTS [14]. We are, in particular, interested in the concepts of local awareness and local decisions.

There are interactive systems other than desktop computers such as large displays which typically use different input devices and, thus, both facilitate and require new and different interaction metaphors than those known from desktops. These arise from their main application in collaborative scenarios. These interaction challenges have been identified and approached by many researchers and several innovative systems and techniques have been developed [6, 7, 8, 9, 16, 17].

Good software requires a careful architecture design which is supported by software frameworks. Designing and implementing an application as a special case of frameworks provides benefits such as modularity, extensibility, and re-usability [15]. Strongly related to frameworks are design patterns which integrate recurring successful solutions to common or standard problems into an architecture [5]. Furthermore, design patterns provide a way to document the structure of a software design, making it more accessible for future users. This is why we chose to provide our software as a framework and to incorporate design patterns.

3. A Buffer Framework Concept

In this section, we outline the overall buffer framework concept before discussing its realization in more detail in Section 4. The buffer framework borrows mainly from three areas: computer graphics, physics, and swarm intelligence which will be introduced in turn.

3.1. Borrowing from computer graphics

The complexity issues with visualization objects and interface components mirror an important scalability trade-off issue in computer graphics: polygonal mesh granularity versus rendering time. Fine-grained meshes produce more detailed and pleasing graphics, but take a long time to render and slow down animations. In computer graphics, layered image buffers were introduced to deal with this problem and to generalize image creation using several properties [13]. We use an approach similar to the rendering buffers in computer graphics for solving the complexity problem in computer interfaces by storing properties of the interface components in a stack of image buffers.

3.2. Borrowing from physics

In most interfaces, interface components are required to interact with other interface components. In order to make these interactions understandable, they have to be guided by the behavior of objects in the real physical world. However, effective interaction does not necessarily follow from closely simulating real physics since it can involve fairly complex correlations that are difficult to understand. In some cases it might be better to derive some kind of interface physics [1] that differs both from real physical interaction and from the approach of limited physics that is used in today's interfaces where only few physical properties are simulated (e. g., non-accelerated straight movement of objects). In fact, interface physics may resemble approaches in non-photorealistic cartoon animation where a special kind of cartoon physics is used that, for example, helps anticipating certain actions [10].

This concept if reflected in the framework by having the possibility to combine two buffers, each containing a specific property, into a single buffer. For example, two separate vector field buffers, each containing movement data for

objects, can be combined into one buffer. Such a combination might be performed by taking ideas from force fields in physics. There, two or more force fields originating from different sources combine to form a single field. In order to achieve understandable interactions between interface components the combination might not always be achieved by simply adding the values of both fields but instead by incorporating the above mentioned interface physics.

3.3. Borrowing from swarm intelligence

Ideas from swarm intelligence are borrowed to significantly extend the number of concurrently active visualization objects. In particular, we model each visualization object as a separate entity with a limited set of parameters and only local awareness. This integrates nicely with the buffer approach from computer graphics since interaction with image buffers in graphics also has a very local characteristic. This allows us to unify and simplify the way visualization objects are interacting with interface components. This results in a significant increase in the number of objects that can interactively be processed. Also, the local character of the interaction with image buffers also allows us to considerably increase the number of concurrent user-interactions with objects or interface components.

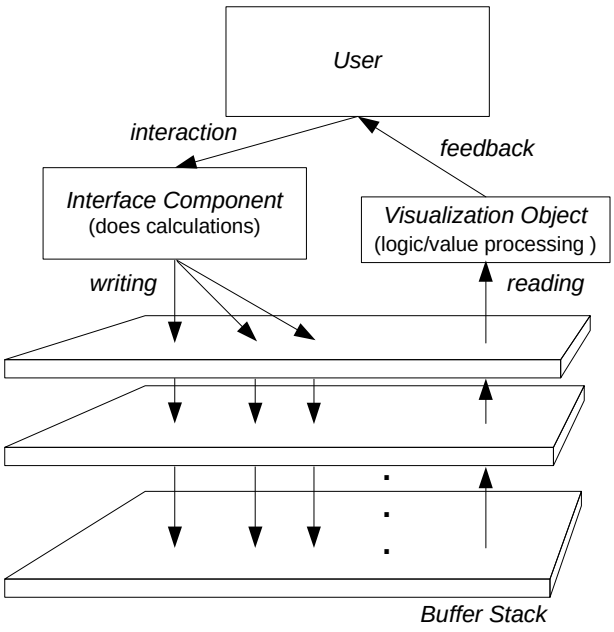

Figure 1. Common relationship of an interface component, a buffer stack, and a visualizaton object during user interaction.

4. Realizing the Buffer Framework

After having introduced the main concept of the buffer framework in the previous section, we will now discuss the components of the framework in more detail and how they interact with each other. In addition, we present some aspects of the software architecture which we used to create a flexible framework that is easy to extend further.

4.1. Framework components and their relationship

As mentioned in Section 1, a typical interface consists of two entities visible to the user: *visualization objects* that are typically carrying the displayed information and *interface components* for organizing and interacting with them. The buffer framework adds a third non-visible entity, the *buffer stack* (see Figure 1). The buffers in the buffer stack are written into by the interface components and thus contain information or data that can be used to control the behavior of the visualization objects. The buffers typically do not contain any logic or algorithmic information but numerical or boolean values. By this means the application logic can be transferred from the interface components to the visualization objects. Interactions and animations of visualization objects are derived from the information read from the buffers rather than from that being controlled directly by the interface components. In this manner, multiple buffers enable very flexible visualizations. The interface components are relieved of the complex administration and steering tasks reducing the computational complexity of the system. By having only local awareness, visualization objects can access from the buffers the data that is relevant to themselves. This data in the buffer is a result of the interface component sampling its properties and writing to the respective buffer location, Visualization objects process their gathered information and act accordingly (see Figure 2). To simplify the collection of the relevant information, each buffer in a stack usually serves a special purpose. For example, there may be buffers for storing object size, orientation, color, etc.; Figures 3 and 4 show examples of these different uses.

The interface components modify the buffers. While the visualization objects typically read and react, the interface components receive input and write to the appropriate buffers in the stack. Thus the interface components indirectly affect the visualization objects as shown in a schematic overview of this process in Figure 1. In this framework, this is the most common configuration for the interaction between interface components and visualization objects, however, different ones are possible as well. For example, one interface component could be controlled by another interface component, i. e., acting as a visualization object and reading data from a buffer instead of writing to it. In addition, a visualization object can also write data to

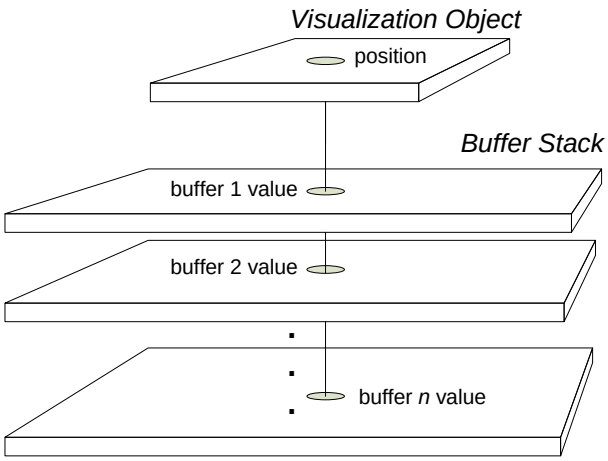

Figure 2. The buffer stack and the local awareness of a visualization object.

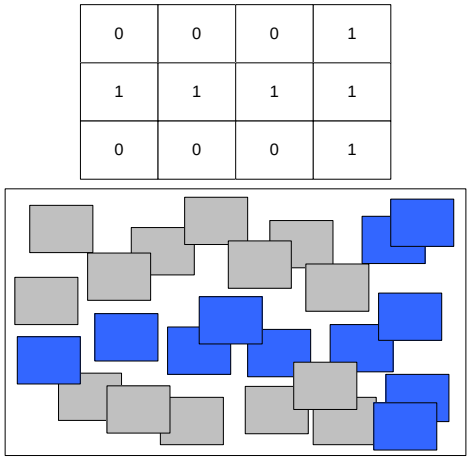

Figure 3. A buffer influencing object color.

a buffer, e. g., to mark its presence in a certain area.

The buffer setup affects both flexibility and performance of the system because the properties stored in the buffers essentially control the entire interface behavior and it is, therefore, of high importance. To model both global properties used by all framework entities and properties local to individual interface components, we provide several buffer stacks. One global stack that covers the visible interface surface and local stacks, one for each interface component, as shown in Figure 5.

These separate local stacks have the advantage of flexible size depending on the size of the interface component or its bounding box reducing the memory requirements of the system. This setup also facilitates the independence of the interface components from each other and allows for

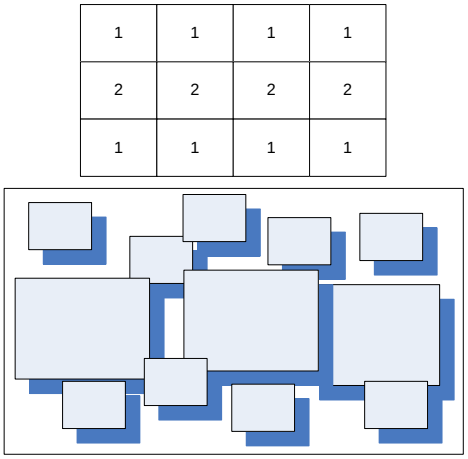

Figure 4. A buffer influencing object size.

greater extensibility. This means that object control is not tied to the global buffer stack size, thus, it is possible to move an interface component partially off the screen while maintaining the integrity of the animation of the object by the interface component.

The global buffer stack is partially used to facilitate associating visualization objects to interface components by providing an ID interface component buffer. After user interaction with a visualization object this ID buffer is used to determine the responsible interface component at the visualization object's new position and the object is attached to the component. The visualization object can then determine the component's buffers it is able to process, links to them, and is consequently affected by their content.

4.2. Software architecture

Any software framework requires easy re-usability in form of a semi-complete but extendable and customizable application [15]. Our architecture fulfills this requirement by relying on object-oriented design patterns [5]. We provide the functionality of a basic and extensible base set while a specific application can be built using a chosen GUI and rendering API. For this purpose we employ the *Builder* and the *Composite* pattern (see Figure 6).

A specific application uses the framework's central *Visualization Control* object for controlling the creation, administration, and rendering of visualization contents. It provides the framework's main loop and coordinates the communication between buffers and the components of the visualization, e. g., its visualization objects and interface components. In order to realize the functionality outlined in Section 4.1, the Visualization Control object also provides functionality to handle multiple inputs, to handle different input, output, and buffer resolutions which are not tightly coupled

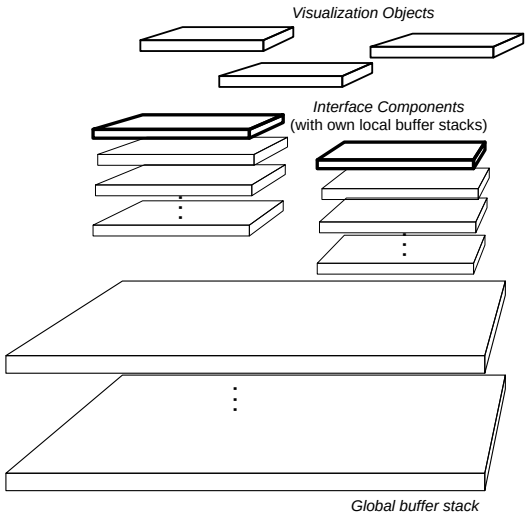

Figure 5. One global and several local buffer stacks.

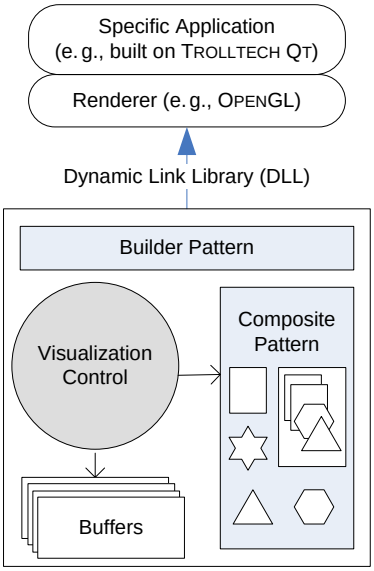

Figure 6. Schematic overview of the framework architecture.

to each other, to change buffer resolutions at run-time, to combine buffers using different methods for realizing the force field metaphor, etc. Multiple concurrent inputs are processed by the main application and then forwarded to the Visualization Control for manipulating buffers as well as objects and interface components.

The Builder pattern provides a unified interface to hide the specific rendering API from the application programmer so that the API can be replaced if necessary. In addition, we also use it to create recurring configurations of components that are commonly used to construct interactive visualizations. These can then be called by an application to create a specific visualization.

The Composite pattern contributes by providing a unifying base for the visualization objects and interface components. These are all treated in a similar way with the distinction that interface components can contain visualization objects as well as other interface components For example, a storage bin [16] may contain many objects such as images but may also contain other storage bins. Both the Builder pattern for visualization components (visualization objects or interface components) as well as the Composite pattern have specific implementations for the chosen rendering API. The implementation of these patterns can be extended to provide new components, however, while the rendering API can be changed, the components would need to be reimplemented to support the new rendering API.

Figure 7 illustrates this concept further by showing the class structure outline for the buffer framework in UML-notation. The buffers themselves are provided by a template class, thus supporting many different types of buffers.

Foundations for the framework are the abstract classes for the Builder, the visualization Control and the Composites. These provide all rendering-independent functionality such as organizing the Composites or processing user input. Subclasses add necessary functionality, e. g., the renderer-specific geometric representation. To use the framework, which comes with implementations for OPENGL, the user would implement a basic application frame with the toolkit of his or her choice. Using the interface of the Builder class the user is able to create a Visualization and its contents, consisting of Composites, which work with the buffers as outlined above. New interaction metaphors are implemented for a specific renderer by sub-classing from the provided abstract interfaces and integrate seamlessly into the framework without affecting existing code. By making the buffer framework independent from any special type of application code, the integration of other toolkits or frameworks into the application is possible. This is especially interesting if considering toolkits for multiple user input, which can then be forwarded through the application into the buffer framework. Figure 8 depicts these interrelations.

5. Applications, Implementation, and Results

Currently, we see the main purpose of this framework as the development of applications for large displays. Sufficient interactive response makes it possible to implement and to evaluate new and complex interaction metaphors. To illustrate our framework's effectiveness, we are in the process of re-implementing a family of 2D tabletop interaction tech-

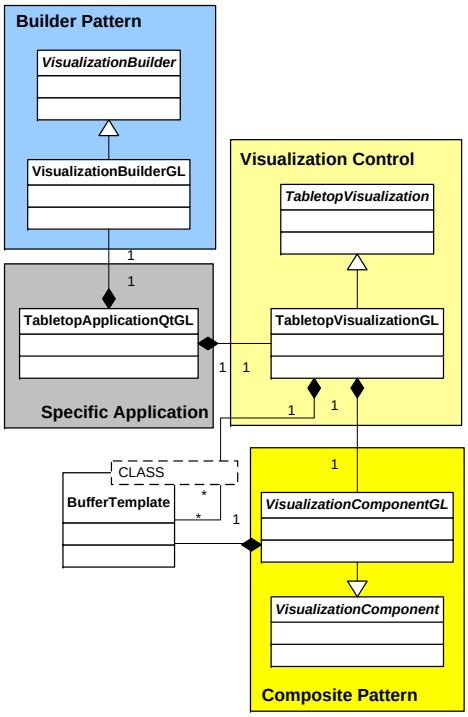

Figure 7. Basic class structure of the buffer framework.

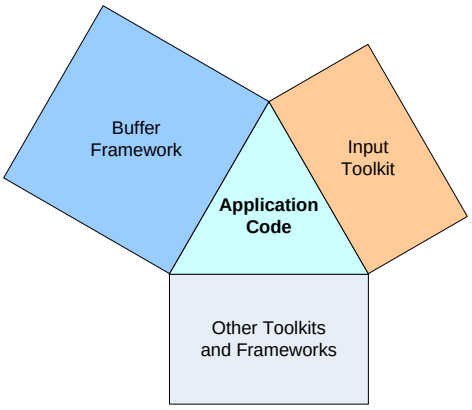

Figure 8. Interrelationship of a tabletop application's typical components.

niques [7, 8, 9, 16] that push the edge of regular methods interactive capabilities. This endeavour has already shown that our framework is capable of reproducing of these complex metaphors and has resulted in combnining several different interface techniques in one unifying application, thus indicating the possibility of creating richer and more powerful interfaces. Also, we are able to maintain responsive animation because the framework uses local awareness of visualization objects and property sampling. These interface metaphors can be represented by a common set of buffers. These are a direction buffer (2D float vector), an orientation buffer (2D float vector or an 1D float with an angle value), a speed buffer (1D float), and a size buffer (1D float). While this is sufficient to build a great variety of metaphors and applications, the framework is not limited to these. Extensions to visualization objects using completely different buffers are possible, without side-effects to existing code and objects.

Our specific implementation of the buffer framework is based on OPENGL as the 3D rendering API and Trolltech's QT as the API for the graphical user interface. Both are exchangeable and can be replaced by other APIs by implementing the respective specific functionality in the software architecture. Using OPENGL and current graphics hardware provides access to common graphics hardware acceleration. In order to make this process transparent for the application programmer, a standard configuration for hardware acceleration is part of the implementation but can be replaced or deactivated if necessary.

In our initial experiments on a $1{,}280 \times 2{,}048$ pixels (≈ 2.6 Mp) tabletop display consisting of two ceiling-mounted projectors (see Figure 9) we have measured a speedup of about one order of magnitude over previous implementations. For example, with the new framework we are able to maintain a rendering frame-rate of about 25–30 fps for 1,000 animated visualization objects (see Figure 10) compared to about 100 objects in a previous application that does not use the buffer framework [7, 8]. We realize that it is difficult to compare applications without hav-

Figure 9. Table setting 1, $1{,}280 \times 2{,}048$ pixels (≈ 2.6 Mp).

Figure 10. Example application with 1000 animated image objects (some of them overlapping) with a look-and-feel and functionality inspired by a previous application without buffer support [7, 8].

ing a full implementation of the functionality in both cases. However, we are currently developing such a comparison application and we think that the speedups seen thus far are promising.

Interestingly enough, we also observed that the screen (or output) resolution, the input resolution, and the buffer resolution can be relatively independent from each other and do not have to match as shown in Figures 3 and 4. In order to maintain a smooth animation even if the buffer resolution is considerably smaller than the screen resolution, we interpolate the buffer cell values based on the area that a virtual buffer cell, centered around the object's position, covers surrounding cells. In our experiments this yielded a smooth animation without the loss of the finer buffer granularity being noticeable. In our current experiments this yielded a considerable reduction in memory consumption without considerable additional processing for the interpolation. However, this may be different for other applications.

In addition, interaction with the animated objects is possible without a noticeable slowdown since each interaction is treated as one processing step among all other object animations. This is due to the relatively small number of interactions when compared to the number of visualization objects which results in as many interactions per second as there are frames rendered. The interaction processing rate can also easily be increased if necessary by having several interaction "interrupts" per object animation loop.

In our current setup we are using a SMART DViT Board from SMART Technologies that provides us with two concurrent and independent inputs. Although they are not identifiable, they provide a natural way of interaction with the tabletop application. The framework's input interface is implemented in a generic way so that we can use other and any number of input devices as well. For example, it would also be possible to implement inputs using several mice attached to a computer using, e. g., the SDG toolkit [20] in order to facilitate debugging and increase versatility of the framework.

We are currently also running experiments with a new tabletop setup that has $2{,}800 \times 2{,}100$ pixels (≈ 5.9 Mp). It consists of four rear-mounted projectors and uses the same SMART DViT Board input (see Figure 11). Even on this

Figure 11. Table setting 2, $2{,}800 \times 2{,}100$ pixels (≈ 5.9 Mp).

setup with more than twice the pixel count, the speedup compared to the traditional implementation without buffers of about one magnitude could be maintained: The traditional setup ran at approximately 5–10 fps while the new prototype achieved about 30 fps for 200 objects, 20 fps for 400 objects, and 7.5 fps for 1,500 objects.

6. Conclusion and Future Work

The major contribution of this paper is a concept for improving the interaction on large displays such as a 12 by 4 foot wall or a 4 by 5 foot tabletop. In these displays, pixel counts may easily reach 9 million and more (compared to 1–2 million in desktops) making it difficult to achieve interactivity with traditional methods. In addition, this concept allows us to more readily support multi-user information exploration on these displays. The unifying framework is based on the integration of several visualization and HCI methods. Together they permit new ways of using rendering techniques to enable communication and comprehension. Finally, by

providing more responsive and locally independent interaction for large tabletop or wall displays we enable the extension of existing interaction methods and provide the possibility for the development of new metaphors.

In the future we envision implementing local object polling by having a local or global object buffer that stores lists of objects present at certain positions. This way objects would render themselves into these buffers and at the next iteration can find all objects in a certain neighborhood. This reduces the search for neighboring objects from $O(n^2)$ to $O(n)$ for a given neighborhood. Since the framework allows us to store any numeric data in buffers we would like to explore using function pointers in our buffers, thus enabling dynamic change in the behavior of objects depending on their position. In terms of new application domains, we will explore using of the framework to develop new ways to present and interact with information, i. e., creating new information visualization metaphors.

In the long run, there are a great number of research questions that will have to be answered as we continue to develop the framework. For example, what does it mean to undo an operation that was mediated by a combined field function, and how is it to be implemented? How many simultaneous interactions can be supported while retaining sufficient interactivity? How well does the framework scale with a growing size of image buffers?

Acknowledgments

We gratefully thank our funding providers Alberta Ingenuity (AI), the Canada Foundation for Innovation (CFI), SMART Technologies, and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] B. B. Bederson and J. D. Hollan. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. In *Proceedings of ACM SIGGRAPH 94*, pages 17–26, New York, 1994. ACM Press.

[2] E. Bonabeau, M. Dorigo, and G. Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[3] E. Catmull. Computer Display of Curved Surfaces. In *Proc. IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structures*, pages 11–17, 1975.

[4] P. Dietz and D. Leigh. DiamondTouch: A Multi-User Touch Technology. In *Proceedings of ACM UIST 2001*, pages 219–226, New York, 2001. ACM Press.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, 1995.

[6] F. Guimbretière and T. Winograd. FlowMenu: Combining Command, Text, and Data Entry. In *Proceedings of ACM UIST 2000*, pages 213–216, New York, 2000. ACM Press.

[7] U. Hinrichs, S. Carpendale, S. D. Scott, and E. Pattison. Interface Currents: Supporting Fluent Collaboration on Tabletop Displays. In *Proceedings of Smart Graphics 2005*, volume 3638 of *Lecture Notes in Computer Science*, pages 185–197, Berlin, 2005. Springer-Verlag.

[8] U. Hinrichs, S. Carpendale, and S. D. Scott. Interface Currents: Supporting Fluent Face-to-Face Collaboration. In *ACM SIGGRAPH 2005 Conference Abstracts and Applications*, New York, 2005. ACM Press.

[9] R. Kruger, S. Carpendale, S. Scott, and S. Greenberg. Roles of Orientation in Tabletop Collaboration: Comprehension, Coordination and Communication. *Journal of Computer Supported Collaborative Work*, 13(5–6):501–537, Dec. 2004.

[10] J. Lasseter. Principles of Traditional Animation Applied to 3D Computer Animation. In *Proceedings of ACM SIGGRAPH 87*, pages 35–44, New York, 1987. ACM Press.

[11] C. Plaisant and A. Rose. Exploring LifeLines to Visualize Patient Records. Technical Report CS-TR-3620, CAR-TR-819, Computer Science Department, University of Maryland, USA, 1998.

[12] R. Rao and S. Card. The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus + Context Visualization for Tabular Information. In *Proceedings of ACM SIGCHI 94*, pages 318–322, New York, 1994. ACM Press.

[13] T. Saito and T. Takahashi. Comprehensible Rendering of 3-D Shapes. In *Proceedings of ACM SIGGRAPH 90*, pages 197–206, New York, 1990. ACM Press.

[14] S. Schlechtweg, T. Germer, and T. Strothotte. RenderBots—Multi Agent Systems for Direct Image Generation. *Computer Graphics Forum*, 24(2):137–148, June 2005.

[15] D. C. Schmidt and M. Fayad. Object-Oriented Application Frameworks. *Communications of the ACM*, 40(10):32–38, Oct. 1997.

[16] S. D. Scott, M. S. T. Carpendale, and S. Habelski. Storage Bins: Mobile Storage for Collaborative Tabletop Displays. *IEEE Computer Graphics and Applications*, 25(4):58–65, July/Aug. 2005.

[17] C. Shen, N. Lesh, and F. Vernier. Personal Digital Historian: Story Sharing Around the Table. *interactions*, 10(2):15–22, Mar./Apr. 2003.

[18] N. Streitz, J. Geißler, T. Holmer, S. Konomi, C. Müller-Tomfelde, W. Reischl, P. Rexroth, P. Seitz, and R. Steinmetz. i-LAND: An Interactive Landscape for Creativity and Innovation. In *Proceedings of ACM SIGCHI 99*, pages 120–127, New York, 1999. ACM Press.

[19] T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics. Modelling, Animation, and Rendering*. Morgan Kaufmann Publishers, San Francisco, 2002.

[20] E. Tse and S. Greenberg. Rapidly Prototyping Single Display Groupware Through the SDGToolkit. In *CRPIT '04: Proceedings of the Fifth Conference on Australasian User Interface*, volume 28 of *CRPIT Conferences in Research and Practice in Information Technology Series*, pages 101–110, Darlinghurst, Australia, 2004. Australian Computer Society, Inc.