

DiamondSpin: An Extensible Toolkit for Around-the-Table Interaction

Chia Shen¹ Frédéric D. Vernier² Clifton Forlines¹ Meredith Ringel³

¹Mitsubishi Electric Research Labs
201 Broadway
Cambridge, MA, 02139, USA
{shen,forlines}@merl.com

²University of Paris 11,
LIMSI-CNRS, BP 133, 91403
Orsay, France
Frederic.Vernier@limsi.fr

³Stanford University
353 Serra Mall
Stanford, CA, 94305, USA
merrie@cs.stanford.edu

Abstract

DiamondSpin is a toolkit for the efficient prototyping of and experimentation with multi-person, concurrent interfaces for interactive shared displays. In this paper, we identify the fundamental functionality that tabletop user interfaces should embody, then present the toolkit's architecture and API. DiamondSpin provides a novel real-time polar to Cartesian transformation engine that has enabled new, around-the-table interaction metaphors to be implemented. DiamondSpin allows arbitrary document positioning and orientation on a tabletop surface. Polygonal tabletop layouts such as rectangular, octagonal, and circular tabletops can easily be constructed. DiamondSpin also supports multiple work areas within the same digital tabletop. Multi-user operations are offered through multi-threaded input event streams, multiple active objects, and multiple concurrent menus. We also discuss insights on tabletop interaction issues we have observed from a set of applications built with DiamondSpin.

Categories & Subject Descriptors: H.5.2
[Information Interfaces and Presentation]: User
Interfaces - *Graphical user interfaces (GUI)*.

General Terms: Design, Experimentation.

Keywords: Tabletop Toolkit.

INTRODUCTION

Even though the idea of computational artifacts for co-located collaboration has been proposed before [15,16], only recently have advances in multi-user touch input and display technologies, such as DiamondTouch [3], SmartSkin [9], and DViT [14], opened up the possibility of new form factors that enable research into face-to-face and shoulder-to-shoulder interactions on direct manipulation surfaces.

Our own research has focused on user interface design and interaction techniques for multi-person tabletop environments [11,12,13]. Tables are a familiar piece of

furniture commonly found in homes, offices, cafés, show rooms, airport and train station waiting areas, and entertainment centers. Tables provide a convenient physical setting for people to meet, chat, look over documents, and carry out tasks that require face-to-face collaboration. Digital documents, however, are commonly used only on single user desktop computers and handheld portable devices, due to a lack of support for face-to-face around-the-table applications.

Making computation disappear into the architectural space is only one of the challenges in the design of a digitally augmented tabletop environment – making the interactions with a digital user interface on the table disappear into and become a part of the human-to-human interaction and conversation is a bigger challenge. In this paper, we describe the unique challenges of multi-user tabletop interfaces, and present a novel toolkit we have constructed, called DiamondSpin, for building tabletop applications. Our goal in creating the DiamondSpin toolkit is twofold: it is meant to allow us to further explore fundamental issues regarding the design of tabletop interfaces, and it is also intended as a toolkit to enable others to quickly build multi-user tabletop applications. DiamondSpin is currently being freely licensed to academic researchers. DiamondSpin provides a real-time polar to Cartesian transformation engine that enables around-the-table interactions. It is a versatile toolkit that allows third parties to develop interfaces for collaborative interactions in ways that are not possible with interactive surfaces today; thus, it allows researchers to address a range of issues, as illustrated by the various projects using it (described in the DiamondSpin Applications section).

MOTIVATIONS AND BACKGROUND

Digitally augmented desks [22] support one-person tasks such as writing, editing, calculating, and drawing. User interface design of a tabletop environment is not merely an extension of desktop systems – people usually sit around a table facing each other as in Figure 1. Simply projecting a conventional user interface onto a horizontal surface would not take into account the unique affordances of tables. Tables predate computers; as such, tabletop user interfaces should preserve many of the familiar and useful properties a physical tabletop affords and allow the natural interaction

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CHI 2004, April 24–29, 2004, Vienna, Austria.

Copyright 2004 ACM 1-58113-702-8/04/0004...\$5.00.



Figure 1. A meeting around a physical table (left) and one around a digital tabletop application that was built using the DimaondSpin toolkit (right).

that people usually carry out on physical tables, e.g., re-orienting objects, passing documents around the table, and spreading and piling documents. None of these requirements are handled by conventional user interfaces. Moreover, when the surface is horizontal, rather than vertical, new interaction metaphors appropriate for tabletops need to be created.

The real need for a toolkit to help push forward the research and development of tabletop HCI became evident through our experience in building the PDH tabletop user interface [12], and from discussions with researchers in various workshops [1,20]. Currently, it is very difficult for a typical programmer or graduate student to create even the most basic interface components that embody some of the fundamental affordances of common tabletop interactions, such as arbitrary positioning and orientation of documents on the table, and multi-person concurrent operations. As a result, many research pursuits are hindered by this lack of support.

As a start, we examined the lessons from our experience in implementing PDH, and the insights we gained from the user study reported in [11]. During the course of designing PDH, we also experimented with and developed visualization and document layout techniques for a circular tabletop [21]. Based on that previous research, we derived a set of requirements for user interface and interaction techniques that we believe are generic and fundamental to many tabletop applications (Table 1).

One of the fundamental challenges we must address in constructing a multi-user tabletop user interface is providing real-time interactivity in the face of multiple simultaneous input events and multiple concurrent object manipulations. Conventional UI software is based on rectangular displays for single-user usage scenarios. A tabletop user interface is for multiple users interacting concurrently and may not necessarily have rectangular views. Thus, many of the display algorithms and image composition algorithms [8] for optimized rendering must be modified, and new architectural support must be designed. The underlying architectural design of DiamondSpin (presented in the next section) is one of the key contributions in this respect.

Table 1. Fundamental tabletop UI functionality.

Requirement	Examples
Visual document management	Arbitrary directional orientation of document placement. Document layout.
Document control and interaction	Visible controls allowing direct document manipulation. Direct passing of individual objects as well as groups of objects among the users around the table.
Manipulation	Bare-hand, stylus and keyboard all should be supported.
Rotational UI	Managing user interface components that are either rotatable or rotation-sensitive (UI components that contain text whose readability would be affected if automatically rotated away from the user, such as a menu bar).
Digital tabletop layout	Creating and managing private and public spaces. Creating and managing multiple virtual tabletops.
Multi-user support	Multiple menu bars. Concurrent multi-user interaction techniques.

THE DIAMONDSPIN TOOLKIT

When designing DiamondSpin, we continuously strove to separate application-related policy issues, which should be left to the application builder, from mechanisms that are basic facilities that DiamondSpin provides. The DiamondSpin toolkit exposes UI functionality through a well-defined API consisting of thirty Java classes and interfaces. In this section, we first describe the architecture that enables the fundamental functionality listed in Table 1. We then present the API and sample application code to illustrate the utility of DiamondSpin.

DiamondSpin Architecture

When multiple people gather around a table, there is no single directional viewing angle or orientation that is ideal for everyone present. At the heart of DiamondSpin is a polar-coordinate system that enables continuous individual document orientation among multiple people with arbitrary viewing angles, as well as the rotation of the entire table surface. Figure 2 shows the underlying architecture of DiamondSpin. Input events are sent to a central tabletop server that contains all the methods that are generic to a tabletop system. These methods implement two engines for: (1) real-time transformation of polar coordinates into a standard transformation matrix for graphics context and input events, (2) handling multi-layer multiple depth display functions, and coordinating multiple threads and tabletop views.

Transformation Engine

In a traditional GUI, it is very common to use a hierarchy of components to subdivide the screen layout into smaller sub-regions. This is possible because in a rectangular interface, a rectangle can be divided into smaller rectangles with each sub-region only operating on a local coordinate system. There is only one common direction of orientation for all displayed visual objects. Unlike an interface based on a Cartesian coordinate system, a polar interface has no predominant privileged direction for displayed documents. There is one and only one center that is meaningful. All the sub-regions must know where this center is at all times.

In DiamondSpin, we have created a framework to describe every element in the display in terms of a polar distance and a polar orientation. Our framework is comprised of two key concepts: (a) Translation of the origin of the conventional Cartesian display (usually at the top left or bottom left corner) to the center of the tabletop display, and (b) Three degrees of freedom (3 DOF) d , α , and β for each element on the tabletop.

d and α are, respectively, the distance to the table center and the orientation angle with respect to this center, while β is the rotation angle of the element with respect to its own rotation point. This rotation point can be its own center, or can be some other point such as the point under the user's finger when she is moving the document. In Figure 3, we illustrate two rectangular elements labeled "Document A" and "Document B" in a polar coordinate space. Since their α angles are different, their labels have different orientations. With the introduction of the 3rd degree of freedom, β , we enable the rotation of every element around its own rotation point, as shown by the darker copies.

To compute the relative position and orientation of each element, the translation from a position (d , α) into a transformation matrix is carried out. Given d , α , and β , our transformation engine carries out the following affine transformation for each document on the table (T = translation, S = scale, and R = rotation, superscript t = tabletop, and e = document element.):

$$T^t_{(\text{width}/2, \text{height}/2)} \circ S^t_{(-1, +1)} \circ R^t_{(0, 0, \alpha + \phi)} \circ T^e_{(d, 0)} \circ R^e_{(0, 0, \pi/2 + \beta)} \circ T^e_{(\text{ElemWidth}/2, \text{ElemHeight}/2)} \circ S_{(-1, -1)}$$

Note that for user input events, we employ the same matrix to compute the inverse transformation that translates the input points from the Cartesian screen space into the tabletop polar coordinate element space, in order to determine the target document for an input event.

The three degrees of freedom, (d , α , β), enable DiamondSpin to support tabletop-specific document handling interactions. For example, DiamondSpin offers a document *passing* metaphor where, in one motion, the user can both *push* and *rotate* a document to the other side of the table.

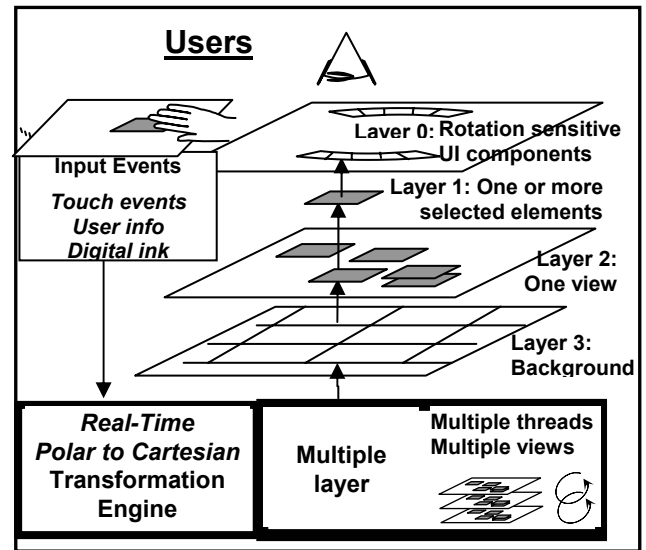


Figure 2: DiamondSpin System Architecture.

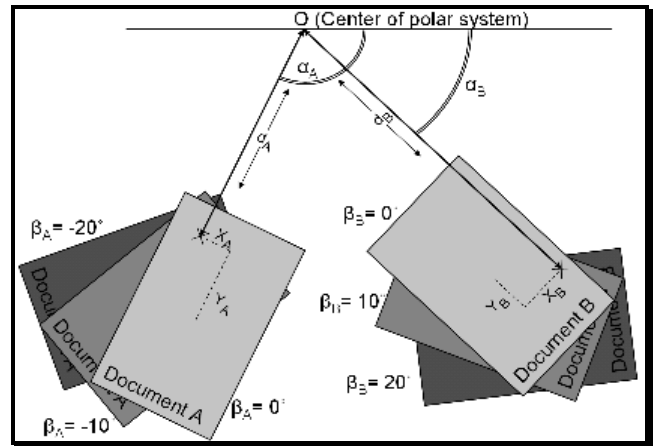


Figure 3: Two documents at distances d_A and d_B to the center O , and at angles α_A and α_B . The rotation points (x_A, y_A) and (x_B, y_B) are used for per-document rotation (displayed at 3 gray levels for $\beta=0$; $\beta=\pm 10$; $\beta=\pm 20$).

We also provide three user-controlled options in our framework for document positioning and orientation: (1) Constrained to face the outside borders of the polygonal tabletop, (2) Facing one common direction, achieving a rectilinear visual layout effect that we call "magnetization", and (3) Freely re-oriented to any arbitrary direction by the user. The first two cases compute β automatically according to d and α . For example, in case 2, the common direction uses $\beta = -\alpha + \phi$ (where ϕ is a global angle used to record the rotation of the entire tabletop). It is also easy to use intermediary values between $\beta = -\alpha + \phi$ and $\beta = 0$ to re-orient documents in a continuum.

Multi-Layer Multi-Thread Display Management

Around-the-table applications put high demand on the display management. As we allow a user to rotate the entire tabletop surface with all of its contents, we must also manage those UI components that should remain rotation

sensitive [4] (i.e., not to be rotated with the entire tabletop surface), such as menu bars. When multiple users drag and drop objects simultaneously, the tabletop interface needs to efficiently manage the display as it refreshes large numbers of pixels. Moreover, because DiamondSpin allows text and image documents to be rotated at arbitrary angles, they must be anti-aliased for quality and readability - a computationally demanding task for an interactive system. Given these tabletop application requirements, sequentially servicing input events will not provide real-time response. Toward this end, DiamondSpin implements a multi-layer representation with a multi-depth multi-thread repaint architecture as shown in Figure 2.

The lowest layer, i.e., layer 3, is composed of non-interactive components (e.g., a background image such as a grid, a map, or a tablecloth texture). Layer 2 is a list of components that can potentially become active. Layer 1 consists of one or more active components. These are the documents or the graphical components that are currently receiving the users' input events, e.g., the documents being passed on to someone else at the other side of the table, or a modal dialog box from which a user is selecting options. To reflect a display change in these components, it is sufficient to merge a refreshed version of them with the other two layers. Finally, rotation-sensitive components, such as menu bars, reside in Layer 0. Each component in a layer is a displayable graphical object with attributes describing its properties.

Together with the multi-layer representation, we also separate the treatment of input event handling and the repaint actions using two independent asynchronous threads. One thread only modifies the parameters of the tabletop element that the current input event is operating on. These parameters are the position of an element (in polar coordinates, of course), the rotation angle of the table, the active view in a multi-view environment, and the size of an element. In parallel another thread repaints the UI at a certain depth according to the input events.

This architecture allows DiamondSpin to selectively refresh part of a tabletop display, thus implementing efficient user interaction schemes and allows an application to use only the layers needed. During most screen updates, only a few actively manipulated components in layer 1 need to be repainted. When a user rotates the entire tabletop surface, we temporarily merge the components in Layers 1 and 2 (but not Layer 0) into one composite texture image to allow the rotation of the entire tabletop in real time.

Discussion of Implementation

DiamondSpin is implemented in pure Java 2D with JAI (Java Advanced Imaging) and JMF. Java is platform-independent, which is crucial for providing a portable and extensible toolkit. DiamondSpin offers mouse input events for conventional input devices. However, the more interesting collaborative activities can only be realized with

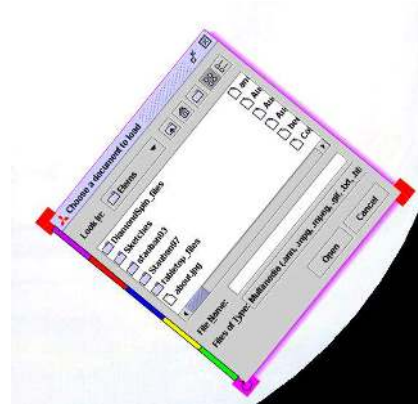


Figure 4: The DiamondSpin framework can display Java Swing components at an arbitrary orientation.

the availability of multi-user input devices. DiamondSpin provides explicit support to run on DiamondTouch [3], a multi-user, multi-touch input surface with unique user identification capabilities.

We have designed a mechanism to transform Java Factory Components into DiamondSpin's polar coordinate system. For example, we have a version of the JComboBox that can be displayed at any orientation, as shown in Figure 4. Rotation of GUI elements in DiamondSpin is achieved through replacement of location functions with polar coordinate computation as well as subclassing of existing Swing Java classes that require graphical popup (DSCombobox, DSMenuBar). In addition we provide a useful method to analyze a tree of UI components and replace unwanted non-rotatable elements by our subclassed components. We leverage off the efficient affine transformation functions that Java provides.

Concurrent user input events are handled by multiple threads, one per user with a unique thread name. DiamondSpin input events are subclasses of Java mouse events with additional fields such as user ID. These extended events are delivered by the toolkit. The toolkit provides the first level of event handling by carrying out the Cartesian to polar coordinate transformation with the inverse matrix described in the Transformation Engine section. This transformation identifies the correct target object on the table for the input event (e.g., a menubar, a frame, or the background.). If the object is a DiamondSpin extended Java element, the MouseEvent is cast into a more generic DiamondSpin event type (e.g., TouchEvent) in order to access those tabletop specific fields such as the user ID. Otherwise, the receiver of the event is a standard Java Swing element (or a Java bean), and it can simply generate component specific callbacks (i.e. actionPerformed() for a JButton).

DiamondSpin API

In this section, we describe a few of the most important classes and then present a sample code segment using the DiamondSpin API to produce a rotated frame.

DSContainer is the main class that embodies most of the functions in the central server described in the last section. It provides methods for input event handling and repaint/refresh of the display, methods to handle document orientation by allowing programmers to specify the angle at which an element should be rotated, and methods to handle “magnetization” which allows the developer to specify a global angle to which all documents should align themselves. The DSContainer also provides the ability to rotate the entire display using the following method:

```
DSContainer.startRotateTable(angle,userID);
```

DSView is the class that is used to create and manage multiple views within the same application. It is also used to create multiple personal and shared work areas within the same virtual tabletop display. A view is an object instantiated by DSContainer. Multiple views allow an application to layout, present, and visualize the same content with a different background. A view receives input events and executes repaint orders from the DSContainer. It provides methods to set a different background image, pan, scroll or rotate the view, and open a contextually appropriate popup menu on an element or on the background.

DSFrame, **DSPanel**, and **DSWindow** are subclassed equivalents of JFrame, JPanel, and JWindow (with or without a titlebar). It replaces location functions with polar coordinate ones and adds angle of orientation and zoom factor functionalities. We also provide optional attributes to be incorporated into each component such as colored shadow generation and finger-size corner handles (as shown in Figure 4 and 5) for resizing or re-orientation. Application developers may use these corner handles plus a shadow as the visual feedback to indicate that a particular document is actively being selected and used.

DiamondSpin provides two types of menus - menu bars and popup menus; they can be used in parallel.

DSMenuBar contains any number of user-defined menu items (icons or text). A menu item can itself be another pull-down menu. As users slide menus around the borders of the tabletop, a pull-down menu often becomes a pull-up menu!

DSPopupMenu, the context-sensitive menu, depends on the location from which it is invoked. For example, a popup menu on a particular document may contain menu items directly related to that document such as a hyperlink selection. A popup menu on the background may contain actions that would affect the layout on the table. A popup menu can be repositioned and reoriented and displays an alpha-blended visual cue connected to the root of its invocation location, as shown in Figure 5d.

Finally, DiamondSpin also offers digital ink with the **DSSStroke** class, and a popup keyboard with the **DSKeyboardPanel** class.

```
DSFrame dsf = new DSFrame("Sample DSFrame", dsc);
dsf.setVisible(true);
dsf.setSize(new Dimension(300, 300));
dsf.setCorners(true);
dsf.setLocation(.4f, (float)-Math.PI/2);
dsf.setBeta((float)Math.PI/4);
dsc.setDragObject(dsf, ID);
dsc.repaint();
```

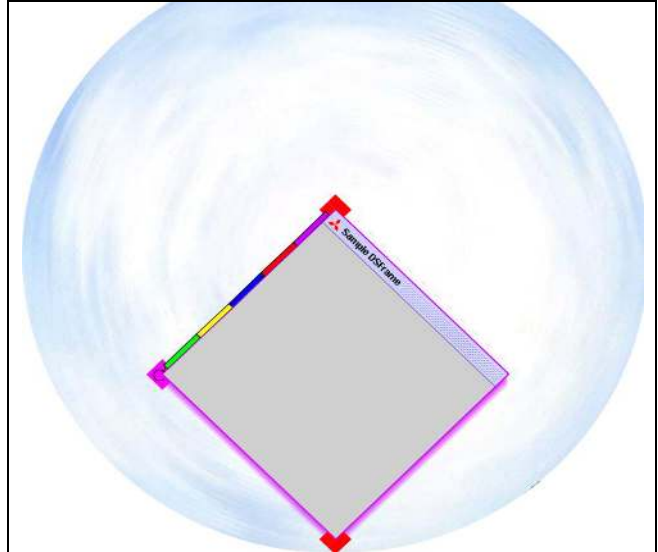


Figure 5. Sample code (top) using the DiamondSpin API to create a rotated frame (bottom).

The sample application code in Figure 5 assumes an instance of DSContainer, dsc, has been created. We first create an instance of DSFrame. The method setCorners() tells the frame to display DiamondSpin's interactive handles to afford rotation and resizing. Next, we choose an initial location for this frame in our polar coordinate system by setting d and α with setLocation() and β with setBeta(). The setDragObject() method makes this frame the current, active object for user ID. Finally, we flag that the DSContainer to repaint. The result of this program is depicted in Figure 5. Now, the DSFrame can be manipulated (moved, rotated, resized, etc.) freely within this view.

In the next section, we illustrate how some of the applications built with DiamondSpin have extended the toolkit's capabilities.

DIAMONDSPIN APPLICATIONS

We briefly discuss five applications constructed using the DiamondSpin toolkit. These applications demonstrate the variety of tabletop user interfaces that the DiamondSpin toolkit can facilitate. Two of the five applications, the Collage & Webpage Builder and the PoetryTable, use surface features of DiamondSpin, and thus were rapidly prototyped within one or two days' time. The other three applications involve more elaborate extensions of the DiamondSpin core classes.

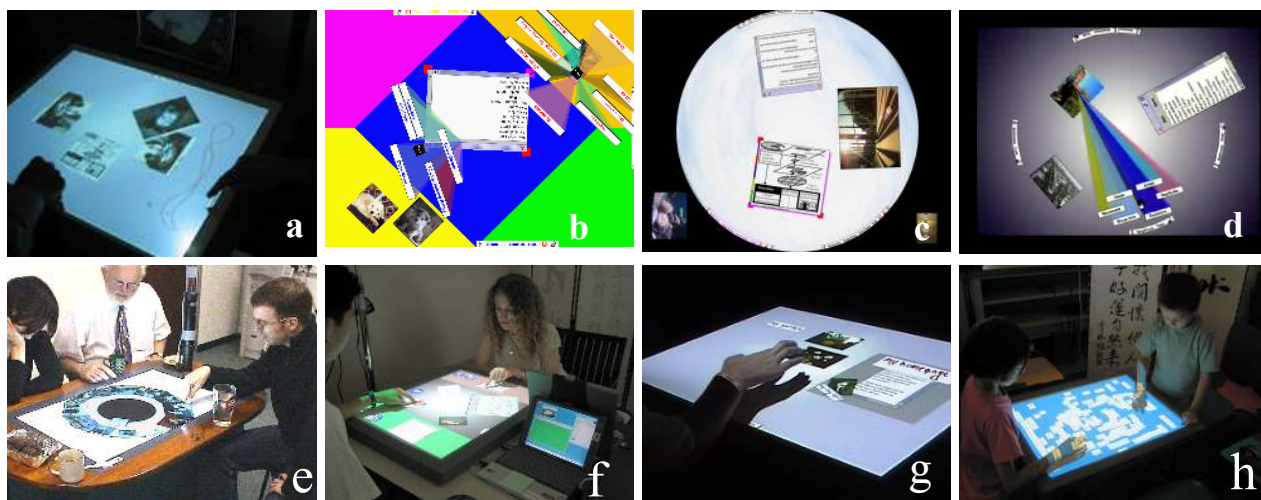


Figure 6. Table for N with (a) rectangular tabletop, (b) a tabletop with 1 shared center, 4 personal work areas, and 2 popup menus, (c) a rotatable circular tabletop, and (d) a rectangular tabletop with continuous document orientation. Other applications include: (e) Opportunistic Browsing Table, (f) UbiTable with laptops associated with personal sides of the table, (g) Tabletop Collage and Webpage Builder, and (h) the PoetryTable.

Table for N

This first application (Figure 6(a)-(d)) is used for a small number of people (two to four) sitting around a table collaboratively creating, sketching, annotating, manipulating, and browsing various types of documents, including text, html, images and video clips. The application extends DSView and provides online dynamic choice among five polygonal views of the tabletop: a rectangular tabletop as “Table for 2” and “Table for 4” allowing 180 or 90 degree re-orientation of contents respectively, an octagonal tabletop allowing a 45 degree re-orientation angle, a rectangular but continuous tabletop allowing 360 degree continuous orientation of contents, a circular rotatable tabletop, and a tabletop partitioned into several distinct work areas.

An Opportunistic Browsing Coffee Table

This is part of an ongoing joint research project with psychologists and HCI researchers on information opportunistic browsing [2] using ambient computing at the Department of Computation of the University of Manchester Institute of Science and Technology, Manchester, UK and Department of E&EE at Imperial College. The environment for opportunistic browsing is a coffee table (Figure 6e) constructed with DiamondTouch [3] hardware and DiamondSpin software. A continuous stream of information items moves slowly along a circular path across the table surface. If an information item (e.g., general news items, local information and announcements) is interesting to a user, he can move it into the center whereupon more detail becomes available for closer examination. A software agent associated with the table senses the nature of the items selected and appropriately modifies the content of the continuous stream of displayed items. The implementation of the coffee table only uses one

DSView. The view handles a list of elements in the circular path. The DiamondSpin application only has to rotate the view through the DSContainer’s setAngle method and add new elements in the view when the table has rotated for a specified angle.

UbiTable

“UbiTable” [13] is a multi-device application built using the DiamondSpin toolkit (See Figure 6f). People can walk up to a UbiTable with their laptops and/or USB devices such as cameras. Users can collaboratively layout, annotate, and mark up content from their own devices on the UbiTable. They can also easily exchange content among laptops and USB devices with others sitting around the table. UbiTable extends both DSContainer and DSView. On UbiTable, each user is provided with a private work area, while the center of the table is a shared area. Thus UbiTable extends DSView to include sub-views. Each sub-view has different types of behavior with respect to document handling and facilities offered. For example, some sub-views have a copier facility, while others contain portals to transfer data to and from laptops.

A Tabletop Collage and Webpage Builder

Collage Builder (Figure 6g) is a direct-manipulation tabletop design and layout application, allowing users to combine images and text at arbitrary sizes and orientations, and to export this work as a web page that uses the collage as an image map linking from each of the collage components to a full-sized version of the corresponding source documents.

The Collage Builder application was created by a graduate student in one day using the libraries of DiamondSpin. The collage is derived from the DSElementGroup class in DiamondSpin, which provides facilities for piling several objects into a single unit. The Collage Builder extends this

class to provide facilities for independent positioning and orientation of the items within the group, and to allow the collage to be saved as a web page.

PoetryTable

PoetryTable (Figure 6h) is an educational game, inspired by the popularity of “magnetic poetry” (<http://www.magneticpoetry.com>). The game allows up to four simultaneous users to combine a set of English or Japanese word tiles to create poetry. The word tiles are automatically rotated to face each side of a rectangular tabletop. Popup menus supported by DiamondSpin give users the option to make duplicates of popular word tiles, to add a suffix or prefix to a particular word, and to save a screenshot of the game in order to preserve their poems. A graduate student created the game over a period of two days using the DiamondSpin toolkit.

One of the most important classes in the PoetryTable game is PoetryGenericView, a subclass of the DiamondSpin toolkit’s DSView, which was extended to include the code for the PoetryTable popup menus (e.g., add suffix, add prefix, etc.). The other major class is PoetryMagnet, a subclass of the DSPanel, which represents the word tiles that are the core of the game.

DISCUSSION OF USER EXPERIENCE

The applications described in the last section illustrate the basic capabilities and extensibility of DiamondSpin. Through the experience of using DiamondSpin to develop applications for touch sensitive, multi-user shared tabletops, many interesting interface design and interaction technique issues have become apparent. The following are a few of them that are part of our current investigation.

Conflicts, even though often unintentional ones, among users are an issue that DiamondSpin has exposed – it is not obvious what should happen if, for instance, one user is interacting with a document while another user chooses to “magnetize” all the documents towards her side of the table.

In a traditional single-user interface, a document is selected when its title bar and frame are highlighted. This metaphor may not translate directly into a tabletop UI – we have observed that users of Table for N need a metaphor that allows them to “drop” (i.e., put down) a document onto the table, while the “selection” metaphor requires them to actively de-select their current object in order to drop it. DiamondSpin’s primitives for indicating which documents are currently active provide a good mechanism enabling further exploration of this issue.

When building applications that use DSPopupMenus to present contextually relevant choices, it is not clear what the best way is to invoke these menus. So far we have experimented with both dwell time and double-tap. This has implications for designing command invocation methods in general on a touch-sensitive surface.

DiamondSpin offers a document *passing* metaphor where, in one motion, the user can both *push* and *rotate* a document across the table. More than one method exists to implement the underlying constraint function of the passing function, resulting in different user experience when a user slides a document around the table. We are conducting user studies to examine this interaction issue.

RELATED WORK

In the past few years, there has been a proliferation of beyond-the-desktop research projects looking at how to integrate the design of computation into architectural spaces and furniture, including tabletops.

DiamondSpin provides support for the class of shared interactive surfaces called Single Display Groupware (SDG), which was first described in [16]. Many previous research projects have examined the design, interaction issues and user experience for various forms of SDG, mostly in the form of shared desktops or whiteboards [5,10,15,17]. DiamondSpin explores interaction techniques for SDG on a new form factor that allows simultaneous multi-person direct touch-based manipulation of information.

The coffee table design in the Living Memory (LiMe) project [7] employs two semi-circle tabletop displays on each coffee table. Each semi-circle displays all the information in a fixed direction. LiMe also explores tangible artifacts. Unlike DiamondSpin, LiMe does not explore tabletop interaction techniques for document manipulation.

The InteracTable [17] and the ConnecTable [18] in the i-Land project supply a rectangular surface to be shared among multiple users in office environments, as well as other office furniture including large interactive whiteboards. The issue of orientation and shuffling of documents is considered very briefly. There does not appear to be support for arbitrary viewing angles, multiple virtual tabletops or subdivided work areas.

DigitalDesk [22] was a physical desk augmented with camera-based vision and projector capabilities so that the physical and electronic desktops are merged into one. DigitalDesk was designed for use by a single person, while our work explores facilitating simultaneous, multi-user interfaces.

MID (Multiple Input Devices) Java Package [6] and SDGToolkit [19] are toolkits created to help building Single Display Groupware. They both offer multiple input devices such as multiple mice or keyboards for a single display. However, in order to facilitate face-to-face interaction where documents can be handled anywhere, at any orientation and in any direction around the table, we must go beyond merely projecting a traditional interface onto a horizontal display, and rotating only the mouse cursors towards the user sitting at a particular side of a

table. In this regard, DiamondSpin is the first vehicle enabling true face-to-face interaction research exploration.

CONCLUSION

We have presented the DiamondSpin Toolkit, including the toolkit's architecture and API. DiamondSpin provides a novel real-time polar to Cartesian transformation engine that enables around-the-table interactions with arbitrary document positioning and orientation on a tabletop surface. Our goal is to preserve the simplicity and informality of around-the-table interaction while supporting tabletop applications for small groups engaged in face-to-face collaborative activities. The approach taken in DiamondSpin has enabled us to design new tabletop interaction metaphors, such as rotating the entire tabletop, thus rotating all the documents within it, the *passing* metaphor that integrates push and rotate into one motion, and laying out documents around the perimeter of the tabletop.

DiamondSpin has proven to be a versatile toolkit to study, build, and experiment with interactive tabletop applications, and to explore open research questions. As we pursue the study of multi-user face-to-face collaboration around the table with DiamondSpin, we will, together with the other users of DiamondSpin, further develop conceptual models and UI components that can be incorporated into the toolkit. The affordances of a digital tabletop are a new territory; we have much exploration ahead of us.

ACKNOWLEDGMENTS

We would like to thank Kathy Ryall, Mike Wu for the many wonderful discussion sessions, Yvonne Rogers for her helpful comments on an earlier version of this paper. We are grateful for all the reviewers' critical comments which have strengthened the paper.

REFERENCES

1. ACM CSCW 2002 Workshop on Co-located Tabletop Collaboration: Technologies and Directions. 2002.
2. de Bruijn, O. and Spence, R., "Serendipity within a Ubiquitous Computing Environment: A Case for Opportunistic Browsing", *Proc. UbiComp 2001*, 362-369.
3. Dietz, P. and Leigh, D., "DiamondTouch: A Multi-User Touch Technology", *Proc. UIST 2001*, 219-226.
4. Fitzmaurice, G.W., Balakrishnan, R., Kurtenbach, G., Buxton, B., "An Exploration into Supporting Artwork Orientation in the User Interface", *Proc. CHI 1999*, 167-174.
5. Guimbretiere, F., Stone, M., Winograd, T., "Fluid Interaction with High-Resolution Wall-Size Displays", *Proc. UIST 2001*, 21-30.
6. Hourcade, H.P., Bederson, B.B., "Architecture and Implementation of a Java Package for Multiple Input Devices (MID)", *HCIL Tech Report No. 99-08*, 1999.
7. Kyffin, S. The LiME Project. *Phillips brochure*. <http://www.design.philips.com/lime/download/brochure.pdf>.
8. T. Porter and T. Duff, "Compositing Digital Images", *Computer Graphics (Proc. SIGGRAPH)*, Vol 18, No 3, July 1984, p. 253-259.
9. Rekimoto, J. "SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces", *Proc. CHI 2002*, 113-120.
10. Russell, D.M, Drew, C., Sue, A., "Social Aspects of Using Large Public Interactive Displays for Collaboration", *UbiComp 2002*. LNCS 2498. 229-236.
11. Shen, C., Lesh N., Forlines C., Vernier F., "Sharing and Building Digital Group Histories", *Proc. CSCW 2002*, 324-333.
12. Shen, C., Lesh, N.B.; Moghaddam, B.; Beardsley, P.A.; Bardsley, R.S., "Personal Digital Historian: Use Interface Design", *Proc. CHI 2001 Extended Abstracts*, 29-30.
13. Shen, C., Everitt, K.M.; Ryall, K., "UbiTable: Impromptu Face-to-Face Collaboration on Horizontal Interactive Surfaces", *UbiComp 2003*. LNCS 2864. 281-288.
14. SMARTTech, "Digital Vision Touch Technology", White Paper, <http://www.smarttech.com/dvit/>. 2003.
15. Stefik, M., Foster, G., Bobrow, D.G., Kahn, K., Lanning, S., Suchman, L. "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings", *ACM Comm*, 30(1), 32-47.
16. Stewart, J. Bederson, B., Druin, A., "Single Display Groupware: A Model for Co-present Collaboration", *Proc. CHI 1999*, 286-293.
17. Streitz, N.A., Tandler, P., Muller-Tomfelde, C., Konomi, S. "i-LAND: An Interactive Landscape for Creativity and Innovation", *Proc. CHI 1999*, 120-127.
18. Tandler, P. Prante., T., Muller-Tomfelde, C., Streitz, N., Steinmetz, R. "ConnecTables: Dynamic Coupling of Displays for Flexible Creation of Shared Workspaces", *Proc. UIST 2001*, 11-19.
19. Tse, E., Greenberg, S., "SDGToolkit: A Toolkit for Rapidly Prototyping Single Display Groupware", *Poster in ACM CSCW 2002*.
20. UbiComp 2002 Workshop on Collaboration with Interactive Walls and Tables. 2002.
21. Vernier, F, Lesh, N., Shen, C., "Visualization Techniques for Circular Tabletop Interfaces", *Proc. AVI 2002*, 257-263.
22. Wellner, P., "Interacting with Paper on the Digital Desk", *ACM Comm*, 36(7), 1993, 86-96.