

Object-Oriented Drawing

Haijun Xia¹, Bruno Araujo¹, Tovi Grossman², Daniel Wigdor¹

¹University of Toronto
{haijunxia|brar|daniel}@dgp.toronto.edu

²Autodesk Research
tovi.grossman@autodesk.com

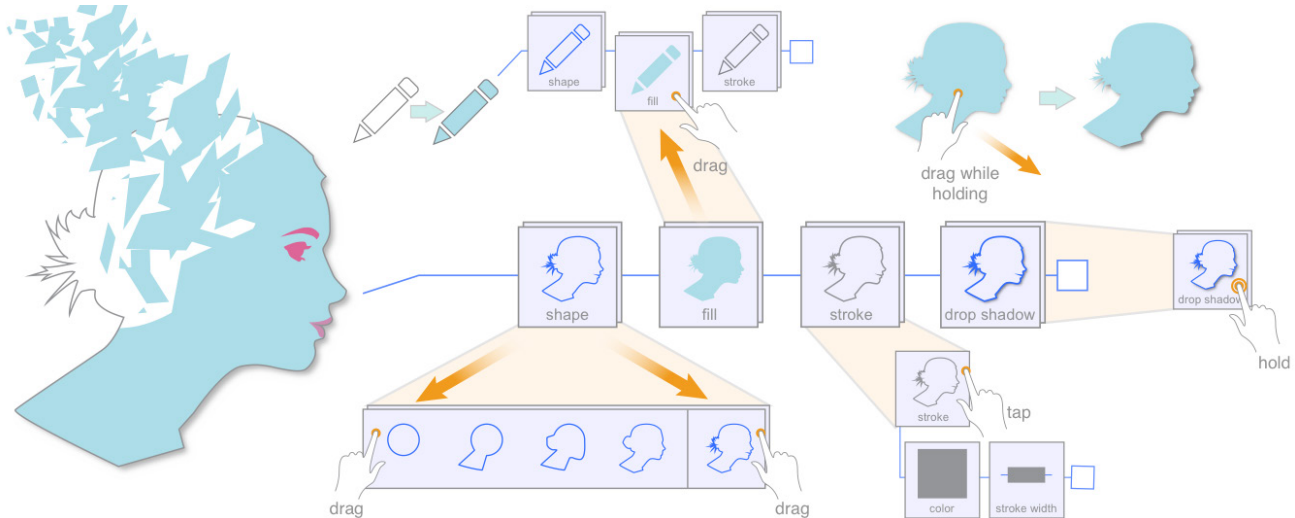


Figure 1. Object-Oriented Drawing replaces most traditional WIMP UI with Attribute Objects which may be directly manipulated with traditional direct-touch gestures. This enables powerful and fluid interaction on touchscreen-based devices.

ABSTRACT

We present *Object-Oriented Drawing*, which replaces most WIMP UI with Attribute Objects. Attribute Objects embody the attributes of digital content as UI objects that can be manipulated through direct touch gestures. In this paper, the fundamental UI concepts are presented, including Attribute Objects, which may be moved, cloned, linked, and freely associated with drawing objects. Other functionalities, such as attribute-level blending and undo, are also demonstrated. We developed a drawing application based on the presented concepts with simultaneous touch and pen input. An expert assessment of our application shows that direct physical manipulation of Attribute Objects enables a user to quickly perform interactions which were previously tedious, or even impossible, with a coherent and consistent interaction experience throughout the entire interface.

INTRODUCTION

In recent years, the popularity of touchscreen devices has exploded, in large part because the omission of dedicated control surfaces allows for bigger screens [20]. As this new input paradigm has gained popularity, so have some

modifications to the traditional Windows Icons Menus and Pointer user interfaces (WIMP UI). Perhaps the single clearest update to the WIMP is the ubiquitous use of direct physical manipulation of content, which can often be rotated, translated, and scaled with simple gestures, rather than utilizing offset controls. Such manipulation in a UI focuses on employing knowledge of the physical world to interact with the digital one [19]. While this leads to interfaces which can be easily guessed, an apparent limitation is that such interaction techniques are limited by their reliance on analogs: the physical world provides no mechanism to directly physically manipulate the brightness or opacity of a photo, nor one to change transition effects between clips of a video. This results in a great deal of variety in the gestures which are guessed by users to try to perform each action, giving clear evidence that there is no universal set of “natural” gestures for HCI [38].

It is perhaps for this reason that applications for touchscreen devices which include even modest levels of functionality often fall-back on the traditional form-filling paradigm which acts as the core of WIMP UI. As an example, consider *Adobe Line* for iPad. Most of the advanced functionality of Adobe Illustrator has been removed, leaving only a few brushes for drawing on a canvas. As Figure 2 illustrates, even in this highly simplified application, we see a regression to the WIMP for control of brush properties. While suitable for a mouse and keyboard, on a touchscreen device, such form filling is tedious, slow, and error prone [8].

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. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

CHI'16, May 07-12, 2016, San Jose, CA, USA
© 2016 ACM. ISBN 978-1-4503-3362-7/16/05...\$15.00
DOI: <http://dx.doi.org/10.1145/2858036.2858075>

The goal of this project is to significantly reduce the reliance on the WIMP UI for touchscreen devices, without requiring users to learn new gestures. We accomplish this by extending the direct physical manipulation metaphor beyond the content of the application, and extend it to the attributes of those objects. We approached this from two directions. First, we sought to explore how traditional WIMP UIs could be replaced with controls which are subject to a direct physical manipulation metaphor. This allows us to provide a UI which leverages and extends the paradigm which has proven successful elsewhere with touch. We hoped this would enable applications with higher levels of complexity without paying the penalty of requiring complex gestural vocabularies. Second, having taken this first step, we sought to understand how enabling such direct physical manipulation could enhance the functionality of a UI to enable a user to quickly perform interactions which were previously tedious or even impossible.

With the UI shown in Figure 2, imagine a situation in which a user wishes to draw first with the “Brush”, then with the “Marker”, both times with the same *size*. Copying the value requires the user to select the first tool, open its properties, note the size value, open the second tool, and manually set the desired *size*. Imagine if, instead, the user could simply grab the *size* attribute as if it were itself an object, and drag-and-drop it onto the “Marker” object. It is precisely this sort of direct manipulation of properties that our project has sought to enable.

In this paper, we present Attribute Objects (AOs). AOs replace much of the traditional form-filling UI by embodying the attributes of digital content as UI objects which can be directly physically manipulated through the same gestures as other primitives. Seeing attributes as more than parameters that define an object’s appearance, layout or behavior, they are treated as components of virtual objects; beyond seeing attributes as abstract numerical values, they are themselves objects which can be directly physically manipulated.

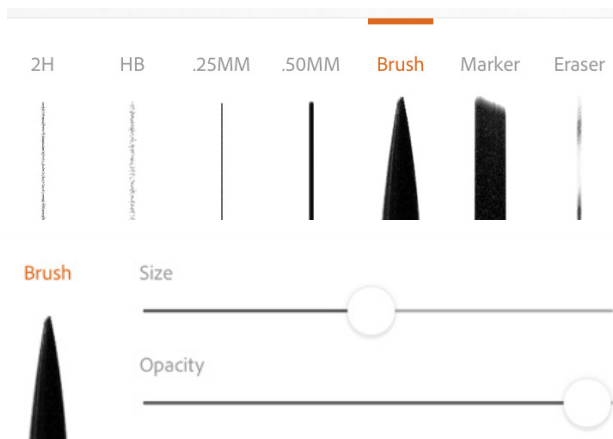


Figure 2. A partial screenshot of Adobe Line for the iPad. Top: the control bar is used to select a brush. Bottom: pressing and holding reveals a control panel of properties for the brush.

To deeply explore the role that AOs can play in a user interface, we developed a drawing application with simultaneous touch and pen input that allows users to directly manipulate the attributes of vector objects. The application as well as the many uses and advantages of AOs, are presented in detail in the paper. We have focused on drawing because it is a rich platform for exploring interaction mechanics, given the genre’s heavy reliance on attributes. That said, it is our intention that the techniques we have developed could be more broadly applied to touchscreen applications of other types as well.

In addition to our interaction techniques, we present the results of an expert assessment, where we asked users with several years’ experience in vector drawing applications to use our application and give feedback on the advantages and limitations of Attribute Objects. We first examine the related work.

RELATED WORK

Our work draws from several areas of previous research: the form-filling user interfaces, objectifying UI elements, pen and touch interaction, touch gestures, and alternative input primitives. We review each in turn.

The WIMP and Form Filling

In their early stages of commercial development, the WIMP UI for the personal computer was regarded largely as a tool for data entry and retrieval [30]. This can be traced back to the Xerox Star system with its property sheet [21], where properties are displayed and changed in graphical forms. To this day, UI toolkits continue to focus on enabling the construction of forms to enable quick and error-free data entry input for transactional human-computer interaction. The research community has long argued for the development of a UI based on what was termed *direct manipulation*, defined in part as “rapid incremental reversible operations whose impact on the object of interest is immediately visible” [30]. Direct manipulation has, for certain classes of operations, become the norm, such as WYSIWYG word processing, models in computer-aided design, and geospatial applications [30]. While direct manipulation is a common paradigm, equally common is a regression to form-based editing of parameters, such as selecting a font from a list or line thickness from a scale. On a desktop, form filling can be fast and precise, thanks to pixel-accurate pointing and keyboard input. On touch-based devices, however, it can be tedious, slow, and error-prone [8]. One relatively recent innovation in WIMP UI is the contextual tool palette, which allows quick access to tools customized to the current selection. We borrow from this in situ placement of attribute controls, and build on it by adding direct manipulation. The goal of the present project is to extend direct manipulation from the *object* of a tool to its *attributes*, and in so doing enable a fully expressive UI that is better suited to touch input.

Objectifying UI Elements

In most form-filling applications, UI controls such as buttons, scroll bars, and menus are the objects that users can manipulate. These objects provide a means to interact with properties, which are mostly text strings or numerical values.

Extensive works have investigated objectifying UI elements to employ knowledge of the physical world to interact with the digital one. However, it is perhaps because of the profound influence of tool use in human history that a number of works have focused on the objectification of tools or commands. For example, controls and tools are no longer plain icons confined in tool palettes; they can be moved around [7, 9, 31] and applied selectively [31]. Different aspects of tool manipulation have also been explored. Bier et al. investigated the bimanual interaction with tools, which were embodied as transparent widgets in a virtual sheet [9]. HabilisDraw [2] focuses on tools as first-class artifacts by importing the ecological properties of tools in physical environment. Nevertheless, manipulation of attributes still has to be delegated to basic controls [6], which affords very limited interactions.

Our work objectifies attributes instead of tools. Direct physical manipulation gestures can be applied directly to Attribute Objects without mediating user’s input through other UI controls. This enables rich and flexible manipulation of attributes. Attributes are no longer numerical values permanently bound with application objects; they can be freely decoupled from one object or attached to another. Besides, promoting attributes to the same level of the application objects also advocates unlocking the global actions to attributes. In our system, attributes can be duplicated to preserve a copy, grouped to form a style, and restored for localized undo.

Pen and Finger Touch Interaction

Significant work has explored the rich interaction vocabulary of touch input, largely divided between touch with a pen (e.g.: [4, 33]), with fingers (e.g.: [26, 34]) and with whole hands (e.g.: [14, 35, 39]). More recently, researchers have sought to expand the capabilities of each by leveraging bimanual, simultaneous use of pen and touch interaction, enabled by hardware that is able to distinguish them [12, 17].

Prior research investigated the complementary roles of pen and touch by assigning pen to the dominant hand and touch to the non-dominant hand [12]. Hinckley et al. instead divided the labor of pen and touch based on the strengths of the input modality: *pen writes and touch manipulates* [17]. Their work introduced a slew of compelling interactions for metaphorical physical manipulation of content and demonstrated the power of such manipulation.

Our work seeks to build-on and extend this, enabling equally powerful manipulation not just of content but of abstract properties as well, thus providing a fully functional UI which reduces the frequency of regression to a form-filling paradigm. Though the principles we have developed may be

applied to other types of input devices, we have focused on touch input, and designed our interaction methods for bimanual use with the pen and finger.

Touch Gestures

Despite its many problems, touch input offers several advantages over mouse input. For example, multiple commands and operations can be phrased into a single gestural action [13], and gestures which make-use of quasi-modes (or “spring-loaded” modes) can reduce error rates thanks to kinesthetic feedback [28, 29]. Furthermore, utilizing gestural commands also reduces recourse to buttons and widgets. By issuing commands through touch gestures, such systems require no additional control area, allowing the screen to be dedicated entirely to content.

While each of these projects shows great promise, research has shown that, for the set of commands common to the WIMP UI, users can only reliably guess a single type of gesture: direct physical manipulation [38]; this likely explains the popularity of physics-based UIs [1, 37]. A common method to increase gesture vocabulary is to provide gesture teaching systems, either as UI widgets (e.g.: [11, 24, 36]) or as a separate tool (e.g.: [5, 11, 16]). Our project is complimentary to such gestural systems, in that the gestures they have developed may, when extended to our techniques, be allowed to manipulate not only objects in the system, but the attributes of those objects as well. In our work, we have not utilized complex gestures, instead we relied on simple tapping and direct-physical manipulation.

Alternative Touch Input Primitives

We seek to develop alternative input primitives and UI mechanisms. Three projects have explored replacing or augmenting tapping selection with crossing-based selection. Sketchpad first introduced pen-based vector drawing and demonstrated dragging, tapping, flicking, and bimanual pen + button gestures [33]. Later, Crossy demonstrated how a pen-based UI could be adapted and improved through the use of crossing, and Moscovich demonstrated the use of crossing for finger touch input [3, 27]. Our project is similar to these, in that we too promote sliding gestures in a touch UI. Unlike these projects, however, which focus on developing new primitives or modifying a UI to suit known primitives, we focus instead on higher-order UIs controlled by traditional finger-touch primitives (tapping for selection and dragging for direct physical manipulation).

Perhaps most similar in spirit to our project is *FlowBlocks*, which replaced the WIMP GUI with directly physically manipulable UI controls, as we have done [10]. The goal of FlowBlocks, however, was to enable a UI for single display groupware on a touchscreen. As such, their UI emphasized adding multiple steps to effect an action. In contrast, our goal is to *reduce* steps and to provide a UI that is more efficient for single-user applications on a touchscreen. The result is a fundamentally different set of user interface controls and interaction methods.

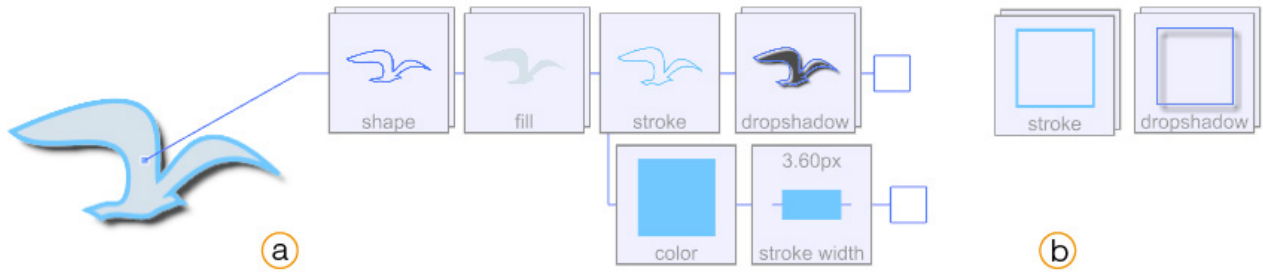


Figure 3. Attribute Object cards in our drawing application. a) Cards have been arranged in a collection, as indicated by their alignment and blue display line. The collection has been associated with a drawing of a seagull; note the connecting line, the square handle for adding new cards, and that the cards show the shape of the gull. b) Free-floating cards sit independently onscreen.

ATTRIBUTE OBJECTS

We propose extending direct physical manipulation, reserved previously for the object of an application, to attributes of those objects. Though simple conceptually, this represents a fundamental overhaul to fundamental UI paradigms. To explore the implications of this change to the GUI, we developed a testbed application where all of the interaction techniques we have described are represented. Though we follow in the footsteps of Bier et al. [9] and Apitz et al. [3] in this use of a drawing testbed, it is our intention that they be considered for general in touchscreen UI.

Identity and Visual Representation

In a WIMP UI, controls such as text boxes and buttons are generally assembled into groupings, such as panes and palettes. Those which represent attributes of an object often indicate those attributes as a state of the widget, such as value of a radio button and contents in a textbox. Further, generic tool palettes exist in a one-to-many relationship with the objects in the system, by changing their values depending on the selected object (e.g.: the “font size” dialogue is attached to the toolbar of a word processor, but its value changes depending on the size of the font in the selected text). In our paradigm, each attribute is assigned an independent identity and is represented as a paper-like card. Cards may be attached to objects, and thus set the value of the attribute for that object, or they may be detached from any object and sit independently on the screen. As an example, each path in a vector graphic has its own Attribute Object representing its *stroke width*, as the shown in Figure 3. The visual representation of the card conveys the following information about an attribute:

Attribute Type: The text at the bottom of the card indicates the type of object the card hosts; for example, stroke, fill, and drop shadow.

Attribute Effect/Value: The effect (or value) of an attribute is encoded by the text on the card and by its dynamic visual representation; see the various attributes associated with the seagull in Figure 3. If an Attribute Object exists on the canvas independently, the card depicts its function on a generic object; see the free-floating cards in Figure 3. If these Attribute Objects are later dropped into a collection associated with a drawing object, their visual representations will update to illustrate that drawing object.

Attribute Hierarchy: A hierarchy of attributes, if one exists, is represented by a tree structure. When it is expanded, it is shown as a second row of attributes, such as in the *stroke* attribute’s *color* and *stroke width* shown in Figure 3. A single paper card represents a base attribute, while a stack of cards—such as the *shape* and *fill* cards shown in Figure 3 indicates a hierarchy which may be expanded.

Direct Physical Manipulation

Attribute Objects cards are objects which can be tapped, held, dragged, and stretched. The positioning of each AO card within the display line can also be rearranged, allowing the user to create groupings, and also change effects of attributes where order matters.

Adding and Removing Attribute Object Cards

In traditional WIMP applications, a control palette lists all possible attributes of an object, including those which have not yet been set. Conversely, with Attribute Objects, an attribute that has not been set simply does not exist. Setting and removing of attributes is accomplished through manipulation gestures:

Removing an Attribute from an Object: Dragging an Attribute Object off the display line removes it from the object. Releasing the attribute on the screen turns it into a free-floating card; dragging it off the canvas deletes it. The attribute is removed from the object, and so it reverts to a state where that attribute is not set. If the attribute is the root of a hierarchy, the whole hierarchy is removed.

Instantiating an Attribute: New attributes for an object may be instantiated by dragging the handle of the display line to the right. As the space between cards expands, potential attributes that can be applied to the object appear; as seen in Figure 4 these are shown as ghostly outlines. A single tap instantiates an Attribute Object and adds it to the collection.

Cloning an Attribute: Attribute Objects may be directly cloned. By holding an AO with one hand and then tapping on the screen with the other, the card is cloned to the tapped position, similar to [17, 32]. A cloned card can then be attached to another drawing object. Cards can also be cloned directly into a drawing object’s collection, simply by holding an AO and tapping the object the user wishes to clone it to. This allows, for example, for the quick and easy copying of style information between graphics objects.

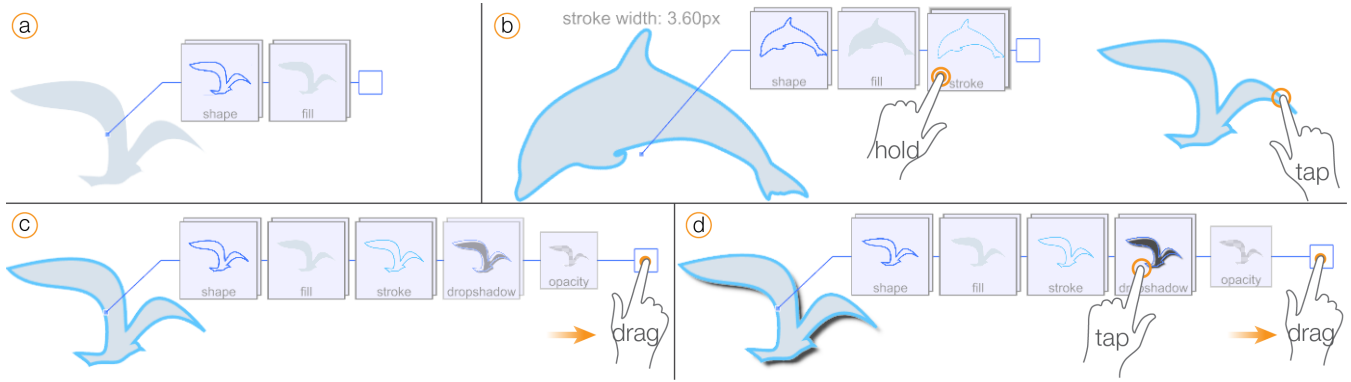


Figure 4. Example interaction sequence: (a) a seagull drawing object contains *shape* and *fill* Attribute Objects. (b) The user clones the *stroke* Attribute Object to the seagull. (c) The user wishes to add a *drop shadow*, so she pulls the handle to see un-set Attribute Objects, and (d) instantiates a *drop shadow* card by tapping it.

Linking

Though a simple clone is useful, additional power of style sharing comes from linking Attribute Objects, so that changes are instantly propagated. Unlike instancing, first introduced in [33], attribute linking allows a many-to-many relationship between several objects' attributes. Attribute Objects are linked at the time of making a clone: immediately after the tap is performed, a link graphic is briefly displayed that, if tapped, will create a persistent connection between the source and destination Attribute Object. Links are shown whenever a linked Attribute Object is manipulated and can be broken by tapping the link icon. Figure 5 illustrates linking.

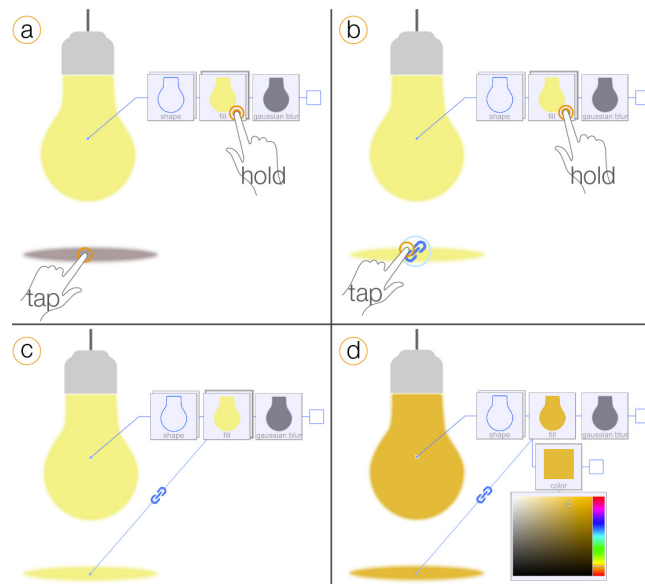


Figure 5. Example of linking: (a) The user has created a drawing of a light bulb and reflection. She clones the fill colour of the bulb into the reflection. (b) When she makes the clone, the *link* icon appears on the reflection – she taps it to link the clone to the source. (c) A link connection appears to indicate the link is established (d) Changes to the *fill* colour of the bulb are continuously propagated to the reflection.

Alignment

Because spatial position is itself an attribute of drawing objects, alignment can be trivially achieved by cloning the vertical or horizontal position of one object to another. Figure 6 shows parts of a crosshair being quickly aligned by cloning and sharing the position attribute; links ensure that the layout is maintained when any object is moved. Cards for each side of an object enables alignment to edges.

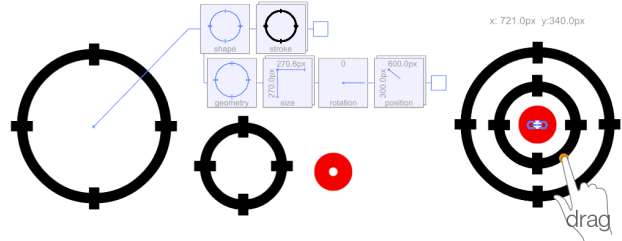


Figure 6. Example of alignment: (a) three separate, circular drawings. (b) the user has cloned and linked the *position* Attribute Object to the other two, instantly aligning all three. The layout continuously is maintained by the links.

Blend

Attributes of the same type can also be blended. This is inspired by the behavior that artists may blend several colors to get the desired one. In order to blend, the user holds two attributes of the same type; this generates a child object from them, taking 50% from each side. The user can slide the child towards one parent or the other, to linearly adjust the influence of each. Dragging a child away from the parents detaches it and turns it into an independent Attribute Object.

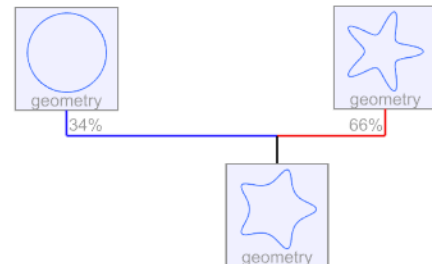


Figure 7. A free-floating star geometry has been blended with a circle geometry to produce a rounded look.

Modes

Attribute Objects can be held to maintain quasi-modes on input, similar to, but more expressive than the ‘shift’ key [28, 29].

Mode of Touch Input: Manipulating Objects or Values

By default, sliding gestures perform standard rotate, translate, and scale manipulation operations. Holding an Attribute Object places the system into a quasi-mode, in which manipulation gestures change the value of the selected Attribute Object. For example, holding the *opacity* card while moving one’s finger on the canvas changes the alpha value; holding a *drop shadow* card while moving the finger directly manipulates the position of the shadow, as shown in Figure 9. This allows users to perform direct manipulations of attribute values. Gestures are mapped to parameters to match user expectations for direct manipulation. As an example, a *rotation* Attribute Object maps orientation relative to the underlying object’s center (similar to a rotation handle), while the value of a *stroke width* Attribute Object is changed by dragging the finger across the stroke.

Such direct manipulation enables us to eliminate most traditional UIs for attribute value selection. The one conceit to a desktop widget is the use of a color picker, since the picker already utilizes direct manipulation.

Mode of Pen Input: Drawing with Attributes

Past work has demonstrated quasi-modes for pen input [17]. In our system, just as holding an Attribute Object places manipulation gestures into a quasi-mode, so too the pen enters a mode related to a held AO. By default, the pen draws a path with its current fill and stroke attributes. If the user wishes to copy the style of an existing on-screen object to a current drawing, they can hold that object (or its desired attribute(s)) with the non-dominant hand. For example, if the user holds an existing *stroke* Attribute Object while drawing, the pen will draw in the style of the touched stroke. Alternatively, if she wishes to quickly copy the shape of an object, she can hold the *shape* attribute and drag with the pen: a new stroke with the same shape will be drawn. See Figure 8.

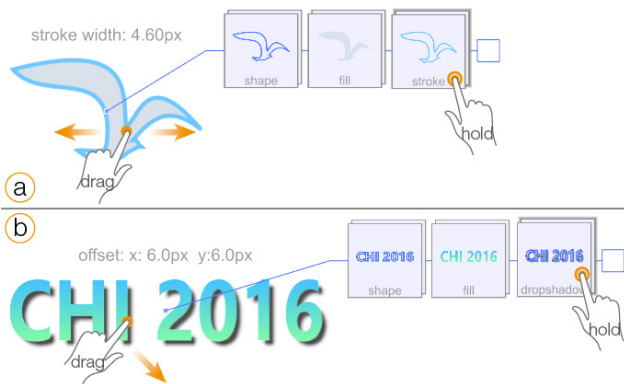


Figure 9. (a) holding the *stroke* card allows the user to directly manipulate the width of the stroke on the canvas by dragging a finger; text appears indicating the current width. (b) holding the *drop shadow* card allows the user to drag directly to change the position of the drop shadow.

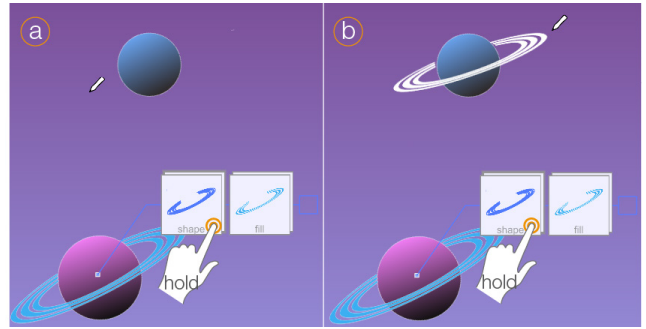


Figure 8. A user modes the pen by holding a *shape* card. This allows him to quickly draw a scaled copy of the planet’s rings. The mode is exited as soon as the user lifts his hand.

History

By promoting attributes to Attribute Objects, there is a natural opportunity to provide a history for card, just as is done for content in many applications. In most applications, *undo* and *redo* commands allow the user to navigate the history of their input: *undo* erases the last change to the virtual content, reverting it to an older state. In a typical WIMP application, the *undo* stack preserves commands executed across the entire application. Therefore, to revert a change to a particular attribute, the user may have to sacrifice all subsequent commands executed on the other objects.

In contrast, each Attribute Object maintains its own history. Instead of tracking commands, we preserve each state of an attribute, not dissimilar to version control applications. As such, a user may retrieve a previous state of any attribute without affecting other objects. In keeping with our theme of embodying attributes, each of the previous states of an Attribute Object is itself an object, with all of the capabilities of other AO cards. It is also worth pointing out that objects of any kind and any level in our system maintains their own history, which enables flexible *undo* across the application.

As an example, a user may decide they liked an earlier color applied to a drawing, and want to use it for a different part of the same drawing. They simply display the Attribute Objects for the original drawing, perform a pinch-to-zoom gesture to expand the *fill* Attribute Object card, and reveal a separate Attribute Object for each of the earlier *fill* attribute values. Tapping a previous state previews the effect. Tapping on the selected state again rolls the attribute back to the state. Histories of parent and child attributes will roll back according to state. Figure 10 illustrates the interaction.

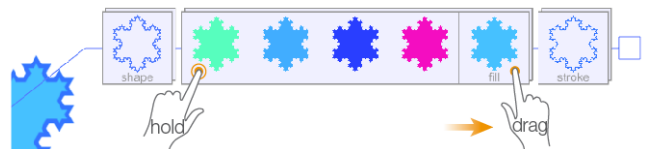


Figure 10. A user expands a *fill* Attribute Object to reveal earlier colors used for the fill. They can then revert, apply those attributes to other objects, or clone the earlier state to a free-floating Attribute Object card for later use.

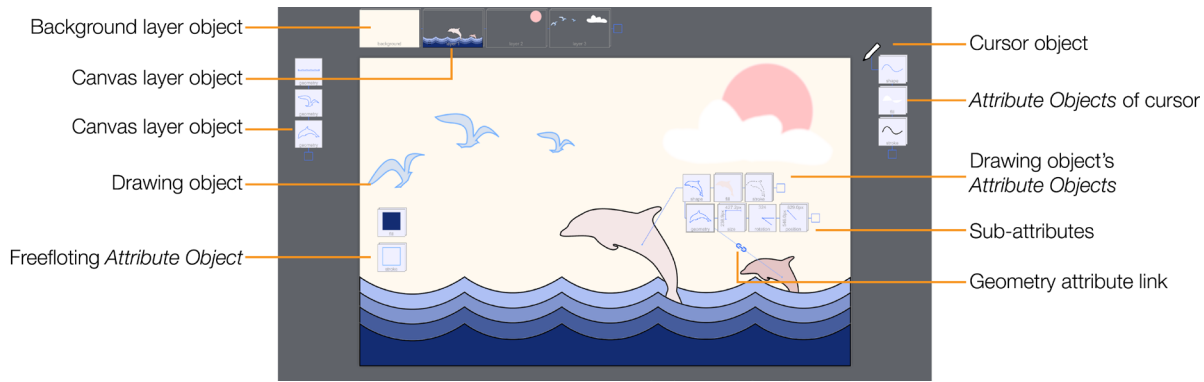


Figure 11. The Object-Oriented Drawing application.

Everything is an Object

We extend our concept of objectification beyond the attributes of content of applications to the controls for those applications, including the cursor and drawing canvas.

Cursor as an Object

While inking, a small pen-shaped cursor provides feedback for user input; similar to the behavior of Microsoft Windows, the cursor remains on the screen when the user lifts the stylus out of range of the digitizer. In keeping with our mantra that “everything is an object”, the user can drag the pointer to reposition it, or tap it to view its attributes. By default, the pen possesses three cards: *shape*, *fill*, and *stroke*. Additional cards may be added or cloned to the pen (see Figure 12).

Being able to draw basic primitives (e.g. rectangle and circle) directly is a desired function. We pre-load such primitives as history states of the *shape* card of the cursor object, enabling the user to easily select these primitives. We made a small tweak to the *shape* card for the cursor object: unlike histories of other cards in our tool, selecting an earlier value will not re-order the list. This ensures access to primitive shapes.

Canvas and Layers as Objects

Just like all other elements of the UI, the canvas and its layers are, themselves, treated as manipulable objects. We treat the canvas as an object possessing a set of Attribute Objects, each representing a layer. By default, there are two layers: the background and a single foreground layer. Each of these

layers possesses a set of Attribute Object cards, each of which may be manipulated just like the others in our UI. For example, the color of the background layer may be changed by directly manipulating its *fill* attribute. The shape of the canvas can be customized by sharing the shape attribute of a path object on the canvas.

Because layers are represented as cards within the canvas, their order can be trivially reorganized by dragging them to another position within the display line. This eliminates the need for dedicated UI widgets (e.g. layer panel of Adobe Photoshop and Illustrator). The opacity of a layer can be adjusted by adding an *opacity* card to the layer that contains the image to support tracing tasks.

EXPERT REVIEW

We wished to validate our belief that further objectifying the UI could add significant value to touch-based systems. We recognize that a fundamental change to UIs could be jarring and, at first, create problems of usability for those who are familiar with a WIMP-based UI. We wanted to ensure that we could gain feedback from potential users about the effectiveness and usefulness of the approach, without being hung-up on initial usability. We thus conducted an expert review with graphic design professionals, each with significant experience with existing drawing tools. We spent significant time training these professionals on the use of the system and collected their feedback on its utility and usability. We were also particularly interested in how these experts would see the concepts integrated into their professional workflows.

Participants

We recruited nine professional graphic designers (4 female), aged 25 to 48, to participate in the review. All participants had more than five years’ experience with a vector graphics drawing tools, such as Adobe Illustrator. Participants were compensated \$50 for an approximately 90-minute session.

Apparatus

The Object-Oriented Drawing system was implemented as a Win32 application in Windows 8 using OpenGL and NVIDIA Path rendering SDK [23], running with a 1920x1200 px Wacom Cintiq 24HD (capacitive finger touch & EMR pen touch + hover). A dual digitizer-based system was utilized to ensure that stylus and fingers could be reliably differentiated.

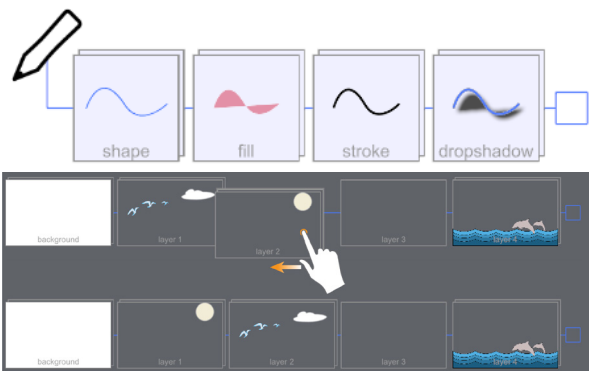


Figure 12. Top: the pen cursor, along with its set of Attribute Objects grouped in a display line. Bottom: Layers are each an Attribute Object of the canvas, allowing for trivial reordering, removal, and other basic operations, without special UI.

Procedure

Each expert review session consisted of three stages:

Introduction and Training (25-30 minutes)

Participants were first given an introduction to the concept. The experimenter then guided the user to explore our interface by finishing a simple drawing of a moon surrounded by stars. During training, the experimenter described the interaction verbally and asked to participants to perform the actions. We intentionally provided only single, simple examples of each interaction technique (such as copying a *fill* color from one object to another, and not copying multiple or root attributes). This was done for two reasons. First, to ensure participants were not overwhelmed. Second, it allowed us to observe whether the participants applied each technique to other contexts, giving them the opportunity to explore and to demonstrate understanding.

Exercise and Freeform Usage (30-40minutes)

Participants were then asked to replicate another simple drawing, provided by the experimenter, which lent itself to many of the techniques, without the assistance of the experimenter. After completing this simple drawing, participants were asked to keep exploring the interface by creating their own illustrations.

Questionnaire & Interview (20-25minutes)

Participants next completed a questionnaire about the system. The questionnaire was composed of 7-point Likert-scale questions to collect the experts' response to both the usefulness and usability of each technique. The experimenter then demonstrated some additional, advanced functionalities of the tool that had not been previously demonstrated but were included in the post-study interview for feedback. The interview then consisted of open-ended questions that were asked to gain the users' feedback on usability, utility, and how well they foresaw integrating AO into their existing toolchain and workflow.

Results

Workflow

Participants were interviewed about how object-oriented drawing could integrate with their existing toolchain and workflow. All responded positively: 5 participants rated their agreement with the statement *"my workflow based on this concept through the entire interface was coherent and fluid"* as "strongly agree", the remaining four "agree". This strongly indicates that the concept can enable graphical applications with a higher level of complexity on a touch-based system, without paying the penalty of inconsistency. Specific feedback from the experts included:

P7: *You think less, and you just like work on graphic more than setting every single attribute. [In Adobe Illustrator], it's so frustrating. Software I normally use, such as Photoshop/Illustrator requires me to set specific settings or use sliders, such as a brush stroke to 1px etc. Your app makes setting attributes more fluid.*

Participants also pointed out that being able to directly access and manipulate the attribute instead of rooting through tools

and menus makes the interface less hidden than the existing WIMP graphic applications.

P2: *In Illustrator there are so many tools and functions hidden, you need to find the tool to change the attributes. In your system, when you want to adjust the attribute, you just do it.*

P5: *There are many things in illustrator that are hidden so deep, and I couldn't discover them. But in your system, I know what attributes I have, and I know what the system is capable of.*

P8: *Tool-wise, you guys don't have the tools; function-wise, you almost have everything.*

Utility of Object-Oriented Drawing

Participants were asked to specifically rate the usefulness of each of the techniques. Significant agreement was found, as shown in Table 1. This indicates that the various techniques enabled by Attribute Objects are valued and desired. A participant noted, *"I love it. I really like the concept that being able to save and share different styles, and being able to access every single attribute of this object, and the history thing, it's just phenomenal."*

Among the various techniques, our system's ability to save and reuse every attribute as well as to preserve the history of each attribute was strongly favored by participants. Participants rated sharing styles in the interface as quick (five "strongly agree", the remaining four "agree") and flexible (four "strongly agree", the remaining five "agree"). In addition, participants reported that saving attributes as floating AOs enabled them to preserve a valuable attribute setting, which may have been arrived-at after significant tuning: *"in Adobe Illustrator, if you screw up the path object, you screw up everything; here I worried less about losing the graphic style"*, noted P2. The statement that "history of attributes provided a flexible way to undo a previous operation from one attribute" was rated "strongly agree" by seven participants and "agree" by the remaining two. The experts also found the object-oriented drawing to be internally consistent and useful for finding functionality:

P3: *The more I used it, the more I enjoyed it. Everything just matches your expectation.*

P8: *Your interface is very nice. It is very uniform. When I want to change something, I know where to find it.*

Technique	Agree	Neutral	Disagree
Creating & Deleting Attribute Object (AO)	9	0	0
Cloning	9	0	0
Injecting	9	0	0
Linking	7	1	1
Grouping independent AOs	9	0	0
AO as mode of touch	7	1	1
AO as mode of pen	9	0	0
History	9	0	0

Table 1. Compressed (from 7- to 3-point scale) summary of Likert-scale responses to "Do you agree this technique is useful for your drawing tasks"

Usability

All participants commented that their interaction throughout the entire interface was intuitive and that the interface was easy to learn. P9 noted: *“here working with my hands like that is really intuitive; all this interaction, I get it. You only have to see it once”*. Four participants reported knowledge and skill transfer from direct manipulation of content:

P2: I can clearly feel each attribute is like an object. The boundary between this tool and the real life disappears. The experience is like how I manipulate things on the chopping board when I am cooking.

P3: It's like drawing in the real world; you are using physical objects. You don't have stroke width in the real world. This is the advantage of the digital software. Your app brought this advantage into an environment that simulates reality, so I can use my experience of the physical world.

Several participants noted that more complex gestures, such as holding and tapping to duplicate an object, and stretching the card to see its history, were not self-revealing. P6 commented *“I wouldn't have guessed that you can see the history, but after I see it, it makes sense”*. This echoes the problem found by many researchers, such as Hinckley et al. [17], and suggests the need for Just-in-Time Chrome or other UIs such as those previously described [36].

Observed Behaviors

We also noted several interesting behaviors during the study. Because the pen is the cursor, we did not enable it to be cloned as an object. However, we observed that one participant (P4) tried to clone the pen object to get several virtual brushes with different styles. This indicates that the participant truly understood the concept that pen is an object. The present alternative in our system is to maintain groups of free-floating Attribute Objects and to use them to provide a quasi-mode for pen input. Of course, this begins to look like a tool palette, though one whose contents are individually manipulable.

Another interesting behavior noted seemed to have been developed out of the understanding that cards are independent objects. Two participants (P3 and P9) developed the drawing strategy where they first worked on the basic geometry of the drawing. They then created independent attribute templates to configure multiple graphical objects, similar to a *style sheet*. Utilizing the attribute copying and linking mechanism, they could quickly configure and adjust the universal styles. P3 and P9 both reported that they developed this strategy because they were *“amazed at the power of quickly sharing and linking attributes”*. The same strategy was seen from P5. However, she reported that in the applications she is using (Adobe Illustrator), she prepares all the styles she will need before she starts drawing. *“It (Object-Oriented Drawing) fits my workflow very well.”*

Usability Concerns

When asked to compare the system with existing vector graphic drawing tools, participants expressed a desire to be

able to precisely manipulate each point of a path. This functionality was omitted to reduce development time but should certainly be included in any commercial release. One participant (P7) also found that sharing attributes in the system is powerful but was, at first, *“a little bit overwhelming”*. P7 noted that *“There are so many possibilities, so what should I do?”* For example, in our system, to share an attribute, users can copy an attribute directly to another object with or without creating a copy on the canvas; they can also hold the object and draw with the pen to re-use the style. Our work focuses on the various interaction techniques enabled by seeing each attribute as an object. However, the comment from P7 indicates that usability might be improved by reducing some flexibility.

Adaptability

In our post-study interview, participants were asked whether they agree this concept can be applied to other applications. All participants rated ‘strongly agree’ and listed the possible applications such as image/music/video editing, layout design, animation, 3D modeling, etc. They also mentioned limitations: e.g. if used for Photoshop, various selection techniques should be supported, so should links between different types of attributes for animation tools.

Summary

The results of the study demonstrate that the interaction methods we have described provide a coherent and consistent interaction experience throughout the entire interface, without sacrificing the functionality of the graphical applications, as is presently done for touch-based devices. Beyond that, more advanced functionalities (such as reusing and linking attributes as well as accessing the history of each attribute) were agreed by our expert assessors to allow a user to quickly perform interactions which were previously tedious, or even impossible. Although the assessment is on a vector graphics tool, our participants found the concept generalizable to various applications.

DISCUSSION

Two core parts form the foundation of our Object-Oriented Drawing application: Attribute Object and the pen + touch interaction. We approached the project from the start as “how do we enable deeper functionality on mobile devices”, and thus much of the design is centered around pen+touch. However, it is clear that the capabilities of Object-Oriented Drawing completely come from the actions that can be applied to Attribute Objects, cloning, linking, blending, etc. The described interactions provide a way to activate such actions on a pen+touch system. That said, the “multi”-touch gestures in the system are cloning, linking, quasi modes, and the drag-to-expand gesture. Each of these has a mouse-based equivalent, such as using “alt-drag” to make clones, “ctrl-click” to select multiple items, and dragging on borders to expand object. Therefore, the concept of Attribute Objects can be easily applied to desktop setup with mouse and keyboard input.

Consider modeling the content in an application and its change with a state machine: attributes describe the states, while commands and tools drive the transitions. Most existing systems are command-centered: users execute commands to make transitions. Our approach, on the other hand, enables direct access to states, with transitions driven by the manipulations of Attribute Objects. Attribute Objects are most beneficial for applications that require intensive formatting (e.g. xml-based content) than data processing (e.g. csv-based content). However, we believe the concept will motivate a new design of data manipulation applications. As an example, the sorting command in Excel reorganizes data. In an attribute-centered design, 'order' could be an attribute of a column. Once attached, the column is reorganized, while the original data is preserved.

It is possible that not all commands and tools could be replaced by Attribute Objects. This reveals one limitation of our system: the limited means (the presented gestures) to drive the transitions. It is a clear area for future work. In our system, holding an attribute invokes quasi-mode. While doing so, it could potentially invoke related tools of the held attribute, for example, the scissor tool for geometry attribute. User can select the needed tool with another hand before manipulation. Holding an AO searches the tools and releasing it clears the modes and tools.

When the number of the attribute becomes large, there is a risk of overcrowding the work area. In our system, the hierarchy and the expandable tree structure of attributes provide a method for increasing the scope of attributes which can be defined per object. Cards can be piled and rogue piles may be moved to a storage area to save the working space. Moreover, object-oriented drawing also advocates a uniform and consistent design of the interface, with inherent clues for navigation in the interface, as pointed out by P8. However, for applications that maintain hundreds of attributes, advanced interface design might still be necessary.

FUTURE WORK

Attribute Objects utilize several properties of physical and digital objects. For example, a physical card can be directly manipulated, while a digital card can be cloned, linked, and can have its state preserved. Other properties of objects can also be imbued into Attribute Objects to enrich interaction possibilities. For example, usage information of an attribute can be conveyed by the crumples developed on the card that houses the attribute, similar to [1, 25]. We also envision that AOs can be hyperlinked with other objects. When a user encounters a desired attribute (e.g. transition effect of a video) online, she can easily clone it. The clone AO is automatically hyperlinked back to the original AO. While using the cloned AO, she can follow the hyperlink back to the original object to get other related AOs.

Another direction of the future work is to further develop the drawing functionality of our test platform to make it a fully functioning vector drawing application. This could facilitate deployment studies and development of a generalized UI.

Beyond this, a more flexible mechanism to enable customized AOs would also be interesting to explore. In our application, users can group attributes to form a style. However, it is unclear what the style is if a group contains color, blur, and offset attributes, even though they are the defining components of a drop shadow attribute. Therefore, to craft a nonexistent AO, the system should allow the users to specify how the effect of an AO is applied. Traditionally, this is achieved by programming. Whether this can be achieved graphically through direct manipulation is a challenging problem and begins to resemble work in end-user programming.

We have explored the interaction with attributes of virtual content in a drawing application on a 2D pen + touch system. Design considerations were made to accommodate this specific setting. For example, an AO is represented as a 2D card which can be dragged and stretched. However, it is unnecessary to keep this metaphor for other applications. For a 3D modeling application in a virtual reality system, a 3D card or ball metaphor might be proper for direct gesture manipulation. Radically, direct tangible manipulation of physically embodied Attribute Objects may also be explored, as the shape changing interface could give dynamic physical representation of virtual content [15].

An AO can be linked to another AO of the same type, so their values will always be equal. Future work can also explore the rich relationship between AO of different types. This can be applied to application for animation editing [22]. In regards to embodying an abstract object to enable direct manipulation, future work can explore embodying the relationship and its attributes as objects. For example, the relationship curve between two variables can be shared with other variables to achieve a consistent animation pace. History of the curve can also be preserved to allow users to retrieve a previous setting.

CONCLUSION

We have presented Object-Oriented Drawing, a drawing testbed which demonstrates the replacement of all traditional WIMP UIs with Attribute Objects. Attribute Objects replace much of the traditional form-filling by embodying the attributes of digital content as objects which can be directly physically manipulated. This enables a drawing application with higher levels of complexity without paying the penalty of requiring complex gestural vocabularies. We have demonstrated through expert review that this approach to touch-based UIs holds considerable promise for enabling more complex functionality on touch-based systems.

ACKNOWLEDGMENTS

We would like to thank members of DGP lab for their inputs and discussions, Katie Barker for editing, and paper reviewers for the valuable feedback. Special thanks to Peter Hamilton for his assistance with the video and Chris De Paoli for the banner.

REFERENCES

1. Anand Agarawala and Ravin Balakrishnan. Keepin'it real: pushing the desktop metaphor with physics, piles and the pen. *CHI '06*, 1283-1292.
2. Robert St. Amant and Thomas E. Horton. 2002. Characterizing tool use in an interactive drawing environment. *SMARTGRAPH '02*, 86-93.
3. Georg Apitz and François Guimbretière. CrossY : A Crossing-Based Drawing Application. *UIST '04*, 3-12.
4. Ronald M. Baecker. 1969. Picture-driven animation. *AFIPS '69 (Spring)*, 273-288.
5. Olivier Bau and Wendy E Mackay. OctoPocus: A Dynamic Guide for Learning Gesture-Based Command Sets. *UIST '08*, 37-46.
6. Michel Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. *CHI '00*, 446-453.
7. Benjamin B. Bederson, James D. Hollan, Allison Druin, Jason Stewart, David Rogers, and David Profit. Local tools: an alternative to tool palettes. *UIST '96*, 169-170.
8. Hrvoje Benko and Daniel Wigdor. Imprecision, Inaccuracy, and Frustration: the Tale of Touch Input. In *Tabletops - Horizontal Interactive Displays*, Springer London, 2010, 249-275.
9. Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony DeRose. Toolglass and magic lenses: the see-through interface. *SIGGRAPH '93*, 73-80.
10. Florian Block, Daniel Wigdor, and Bc Phillips. FlowBlocks: A multi-touch UI for crowd interaction. *UIST '12*, 497-507.
11. Andrew Bragdon, Robert Zeleznik, Brian Williamson, Timothy Miller, and Joseph J. LaViola, Jr. GestureBar: improving the approachability of gesture-based interfaces. *CHI '09*, 2269-2278.
12. Peter Brandl, Clifton Forlines, Daniel Wigdor, Michael Haller, and Chia Shen. Combining and measuring the benefits of bimanual pen and direct-touch interaction on horizontal interfaces. *AVI '08*, 154-161.
13. Bill Buxton. Chunking and phrasing and the design of human-computer dialogues. *IFIP '86*, 475-480.
14. Xiang Cao, Andrew D. Wilson, Ravin Balakrishnan, Ken Hinckley, and Scott E. Hudson. ShapeTouch: Leveraging contact shape on interactive surfaces. *TABLETOP '08*, 129-136.
15. Sean Follmer, Daniel Leithinger, Alex Olwal, Akimitsu Hogge, and Hiroshi Ishii. inFORM: dynamic physical affordances and constraints through shape and object actuation. *UIST '13*, 417-426.
16. Dustin Freeman, Hrvoje Benko, Meredith Ringel Morris, and Daniel Wigdor. ShadowGuides: Visualisations for In-Situ Learning of Multi-Touch and Whole-Hand Gestures. *ITS '09*, 165-172.
17. Ken Hinckley, Koji Yatani, Michel Pahud, Nicole Coddington, Jenny Rodenhouse, Andy Wilson, Hrvoje Benko, and Bill Buxton. Pen + touch = new tools. *UIST '10*, 27-36.
18. Raphaël Hoarau and Stéphane Conversy. Augmenting the scope of interactions with implicit and explicit graphical structures. *CHI '12*, 1937-1946.
19. Robert J.K. Jacob, Audrey Girouard, Leanne M. Hirshfield, Michael S. Horn, Orit Shaer, Erin Treacy Solovey, and Jamie Zigelbaum. Reality-based interaction: a framework for post-WIMP interfaces. *CHI '08*, 201-210.
20. Steve Jobs. 2007. Mac World Expo 2007: Keynote.
21. Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard, and Kevin Mackey. The Xerox Star: A retrospective. *Computer* vol. 22, no. 9 1989, 11-26.
22. Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. Kitty: Sketching Dynamic and Interactive Illustrations. *UIST '14*, 395-405.
23. Mark J. Kilgard and Jeff Bolz. GPU-accelerated path rendering. *ACM Transactions on Graphics* 31, 6: 1, 2012.
24. Gordon Kurtenbach and William Buxton. User learning and performance with marking menus. *CHI '94*: 258-264.
25. Justin Matejka, Tovi Grossman, and George Fitzmaurice. Patina: Dynamic Heatmaps for Visualizing Application Usage. *CHI '13*, 3227-3236.
26. P McAvinney. The Sensor Frame-A Gesture-Based Device for the Manipulation of Graphic Objects. 1986.
27. Tomer Moscovich. Contact area interaction with sliding widgets. *UIST '09*, 13-22.
28. Jef Raskin. *The Humane Interface: New Directions for Designing Interactive Systems*. 2000.
29. Abigail Sellen, Gordon Kurtenbach, and William Buxton. The prevention of mode errors through sensory feedback. *Human-Computer Interaction* 7, 2, 1992, 141-164.
30. Ben Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour & Information Technology*. 1982.
31. Randall B. Smith. Experiences with the alternate reality kit: an example of the tension between literalism and magic. *CHI '87*, 61-67.
32. Sven Strothoff, Wolfgang Stuerzlinger, and Klaus Hinrichs. Pins 'n' Touches: An Interface for Tagging and Editing Complex Groups. *ITS '15*, 191-200.
33. Ivan E. Sutherland. Sketchpad a man-machine graphical communication system. *SHARE design automation workshop*. 1964.

34. Pierre Wellner. The Digital Desk Calculator: Tactile Manipulation on a Desk Top Display. *UIST '91*, 27–33.
35. Daniel Wigdor, Hrvoje Benko, John Pella, Jarrod Lombardo, and Sarah Williams. Rock & rails: extending multi-touch interactions with shape gestures to enable precise spatial manipulations. *CHI '11*, 1581–1590.
36. Daniel Wigdor and Dennis Wixon. Brave NUI World: Designing Natural User Interfaces for Touch and Gesture. 2011.
37. Andrew D Wilson, Shahram Izadi, Otmar Hilliges, Armando Garcia-Mendoza, and David Kirk. Bringing physics to the surface. *UIST '08*, 67–76.
38. Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. User-defined gestures for surface computing. *CHI '09*, 1083–1092.
39. Mike Wu and Ravin Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. *UIST '03*, 193–202.