Tangible Actions

Dustin Freeman and Ravin Balakrishnan
Dept. of Computer Science
University of Toronto, CANADA
{dustin|ravin}@dgp.toronto.edu

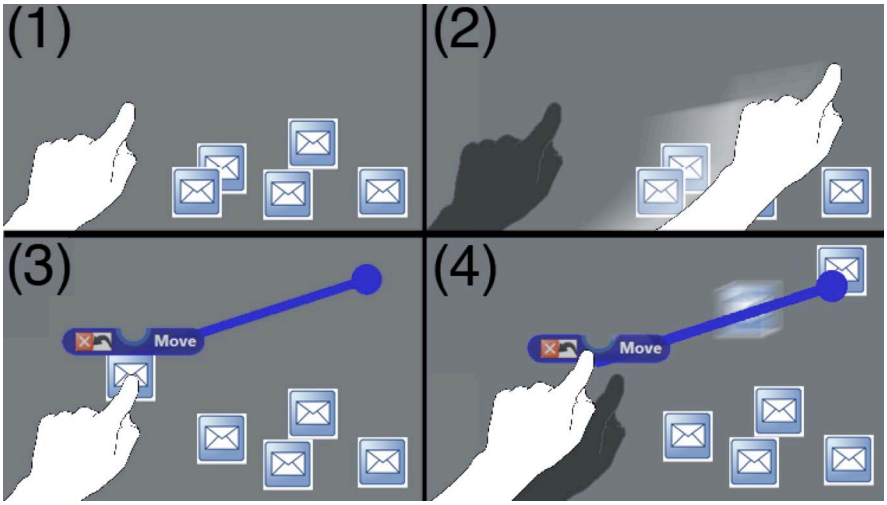

Figure 1. The process of creating (1,2) and using (3,4) a Tangible Action. The user performs a "move" gesture by dragging an imaginary object from left to right across the background (1,2). The "Move" Tangible Action appears when they lift their finger off the interface (before 3). The user can drag objects to the activation point of the Tangible Action (3), and release them to apply the action to the object (4). The user no longer needs to perform the "move" gesture for each object.

ABSTRACT
We present Tangible Actions, an ad-hoc, just-in-time, visual programming by example language designed for large multitouch interfaces. With the design of Tangible Actions, we contribute a continually-created system of programming tokens that occupy the same space as the objects they act on. Tangible Actions are created by the gestural actions of the user, and they allow the user to reuse and modify their own gestures with a lower interaction cost than the original gesture. We implemented Tangible Actions in three different tabletop applications, and ran an informal evaluation. While we found that study participants generally liked and understood Tangible Actions, having the objects and the actions co-located can lead to visual and interaction clutter.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Multitouch, Gestures, End-User Programming, Programming by Example, Visual Programming, Scripting

ITS 2011, November 13-16, Kobe, Japan.

INTRODUCTION

The progression from command-line interfaces (CLIs) to graphical user interfaces (GUIs), to even newer patterns such as so-called Natural User Interfaces (NUI) [28] represents a motion from linguistic to manipulative input as the primary activity of the user. We view Tangible Actions as an extension of direct manipulation that re-examines the status-quo object-action relationship seen in modern GUIs. Traditionally, to apply an action to an object in a GUI, we would "select" that object, or group of objects, and trigger the action elsewhere, such as a toolbar. Instead, Tangible Actions occupy the same space as the objects they act on, and since they are created by user gestures and support basic scripting features, they represent a simple visual programming-by-example language. This has several useful implications for large, multi-user interfaces.

How Tangible Actions work: "Move" Example.

See Figure 1 for an example of a Move Tangible Action. The user performs an "abstracted" move gesture by touching the background of the interface with a single finger (Figure 1-1) and moving a short distance left to right (Figure 1-2). "Abstracted" because it is not performed on an object, but because it is performed on an imaginary object in empty space. This creates an iconic representation of the: a "Move" Tangible Action, which behaves in a variety of interesting ways. If an object is passed over the point where the user's move gesture started, which we call the activation point, it jumps to the end point, and a transition

animation appears showing the performance of a Move action (Figure 1-4). The Move gesture is reduced to the action of moving an object from the beginning point to the end point, without any details of how the user themselves moved, similar to Interactive Beautification [18]. The Move gesture can be continually "replayed" by placing objects at the activation point. Only literal playback of a gesture would be very limiting, so Tangible Actions can also modified. The user can modify both the activation point and the end point by dragging them around like any other interface object. If the activation point is dragged over an object, the Tangible Action is automatically applied to that object.

Extending Direct Manipulation

In direct manipulation, users do not have to determine the linguistic command for a desired action, but may simply attempt to perform it and evaluate the result. In multitouch interfaces, the breadth of interactions available by direct manipulation is much larger. With manipulation-based interaction, rather than language-based interaction as in a CLI, novice users can start quicker and expert users can perform simple tasks more easily. It is not surprising then, that as interfaces are designed to appeal to more and more users, they increasingly favour manipulation. However, by using this style of interaction, we have lost many of the valuable properties of CLIs.

Without linguistic input, the user is always stuck at the novice level of expressivity, and is unable to perform their tasks more quickly, with abstraction or automation. There is no flexibility as seen in CLIs, which allow a smooth transition to reuse and modification of previous commands, and even scripting. Perhaps one reason why CLIs are so flexible is that it is obvious to the user that every action they make clearly maps to a system command. In a GUI, it is not clear that every user action is a command, and thus there has been no way to fluidly create ad-hoc scripts in the same mode as performing manipulative actions. To novice users, it may not even be clear that automation is possible.

We hope that referring to previous actions can be like a conversation, where people can agree on a temporary alias for an idea and can then refer to it in short-hand. In conversation, we often casually refer to earlier linguistic statements or physical gestures [11]. While we have the opportunity to refer to previous linguistic input in a CLI, the same ability does not exist for manipulative input as it does with conversational gestures in the real world.

We want the desirable properties of direct manipulation in GUIs, without losing the desirable properties of linguistic input in CLIs.

TANGIBLE ACTIONS CONCEPT

We define Tangible Actions as *continually-created, interactive in-situ, visual representations of the user's interface actions that can be modified and reused at a lower interaction cost than re-performing the original interactions.*

Continually Created

A Tangible Action is created every time the user performs a meaningful action on the interface. Unlike some previous PBE systems, Tangible Actions does not make an attempt to infer user intent. For the purposes of this work, we aim to demonstrate the usefulness of the principle of application-agnostic Tangible Actions. If an application is more constrained, then it may be easier to infer intent and present more useful Tangible Actions.

In-Situ

After a user performs an action, a Tangible Action is created and displayed in-situ. This is so the user is made aware of what action they have just performed, and can reuse it without having to switch contexts. A user interface action only makes sense as a Tangible Action if it can be performed and represented in-situ. Thus, actions like "Restart" or "Change screen brightness to 75%" are not suitable.

Visual Representations

We represent Tangible Actions visually instead of textually, as they occupy the same space as the visual objects they act on. We provide affordances to show the "activation point", connecting points for other Tangible Actions, and areas that gestures can be applied to to modify the parameters of the Tangible Actions.

Modification and Reuse

The primary purpose of implementing Tangible Actions is so that actions can be reused. It is limiting to allow only literal replay of an action, so Tangible Actions are modifiable and re-usable in different context that the one in which they were created. While prototyping Tangible Actions, we found that the most useful Tangible Actions *act on objects*. Since the usefulness of a Tangible Action is to be applied to multiple objects, the semantics of the object should be easy to generalize; the parameters of the type of action should not have to be customized for each object. An example of a good candidate for a Tangible Action is a move action, or a resize action. A bad candidate is a rename action, as surely the user doesn't want to rename several files the same name. Any action that would be applied to a group of objects could be turned into a Tangible Action. Most actions available by direct manipulation fall into this category, although some, e.g. crop, may be too specific.

Low Interaction Cost

The value of a Tangible Action is that it may be reused with multiple objects, without the interaction cost of re-invoking the action each time. We suggest there may be a tension between the feel of "expressiveness" of an interface and the efficiency of an interface; an interaction that feels expressive is inherently inefficient. Thus, newer interfaces that emphasize manipulative interaction may feel more expressive but are less efficient. As we noted before, Hutchins, Hollan and Norman said that the feel of directness is enhanced when the user specifies an action on an object of interest by mimicking it themselves [17]. However, the mimicking actions we perform in manipulative interfaces require high energy. The current alternative seems to

be the relatively indirect strategy of selecting the object and invoking the action elsewhere, such as in a menu. Tangible Actions can solve this as the user manipulates the Tangible Action minimally to apply the action to the object, yet still has the feeling of directness as their action occurs in-situ, and the action itself is visually represented.

RELATED WORK

Tangible Actions represents an extension of the ideas behind direct manipulation. We summarize previous work on direct manipulation and then discuss work on creating programs by example, in-situ interaction, visual programming, reuse of user actions and decreasing interaction cost.

Work on Direct Manipulation

Shneiderman introduced the concept of Direct Manipulation as "beyond" programming languages [32]. Beaudouin-Lafon describes an interaction model that aims to generalize the principles of direct manipulation [2]. He states that interaction between the user and domain objects is mediated by interaction instruments, analogous to tools and instruments in the real world acting on physical objects. Bødker and Andersen present an even more thorough analysis of the semiotics of the use of interaction instruments in Complex Mediation [8]; there is a similarity between Tangible Actions and interaction instruments.

Creation of Programs by Example

There have been many Programming by Example (PBE) systems, some using the term Programming by *Demonstration*. The simplest is a macro recorder. This work is similar to macros with their notion of record and replay, except that instead of invoking the macro by a menu selection or shortcut, the macros are invoked immediately on objects when an object hits a Tangible Action's activation point.

Myers called *Demonstrational Interfaces* systems where the user creates automation of a task by performing it [27]. The hope is that a PBE system to be able to create a program based on observing the user's interaction and using inference to determine intent. This ability is conspicuously absent from any widely-used interface over 20 years later, so we assume inference is difficult and do not attempt it in our system. Myers says that a system is only a Programming By Example system if it is a Demonstrational Interface and it is also programmable: "I use the term *programmable* for systems that can handle variables, conditionals, and iteration" [25]. We note here that, by this definition, Tangible Actions is not, at the moment, programmable.

During the course of a task as a user transitions from performing simple actions, to re-using some previous actions, they are effectively starting to "program". There has been a large amount of work on *explicit* PBE, where the user must anticipate the need for automation or a program in advance. The user must explicitly begin "recording" their actions, or enter a separate programming mode. Potter identified a set of obstacles a user encounters to actually using explicit PBE, which he collectively termed as the *Just-In-Time programming problem* [29]. He suggests that this is why explicit PBE is not widespread.

Ruvini [31] suggested *implicit* PBE, where the system continually listens to the user's actions, and offers suggestions of automation. Ruvini identifies the two main issues with implicit PBE as learning what to automate and learning when to make a suggestion. However, a PBE system should not be only implicit: if the user perceives a task needs automation, they should not have to repeat it twice before the system offers to automate it. Ideally, users only do anything once! From surveying previous work, we identify two major obstacles to PBE: reducing distraction caused by useless suggestions of repetitive actions and allowing explicit and implicit PBE.

Watch What I Do: Programming by Demonstration [12] and its follow-up, Your Wish Is My Command: Programming by Example [23] provide good coverage of the space of PBE. However, none of the systems describe using manipulations as programming tokens, or represent the programmable actions in-situ with the objects themselves.

In-situ Interaction

A novel feature of Tangible Actions is that it represents interaction history in-situ, with the intention of affording further interaction and modification of the interaction history. Su et al. display an overlay of an interaction history in a drawing program when history is invoked [35]. As individual actions in the overlay may be selected and undone, it is similar to Tangible Actions. The goal of the system was to aid users in building complex illustrations by providing a selective undo mechanism in context, rather than a disconnected list view. Discussing interaction history with respect to individual interaction tokens brings to mind Edwards et. al.'s discussion of managing undo and redo for separate objects as a complex time graph [13].

It could be said that using Tangible Actions represents an *end-user customization* of the interface. This is especially true of the Tangible Actions that are dragged off the top toolbar (Figure 2); this is notable as the customization is not a one-off activity, but happens continually. There may be multiple instances of "toolbars" that move around for the users' convenience, independent of the UI designer's initial ideas. This idea has appeared before in Stuerzlinger et al.'s UI Facades [34].

Visual Programming

Tangible Actions uses of affordances to indicate programming tokens can be snapped together. This was previously used in the Scratch visual programming language [24]. There are some examples of programming by example systems where the resultant program is expressed visually in some way. However, none of the programming representations are in-situ. Kurlander and Feiner present a Macro By Example system that represents actions in a comic strip format [22]. Sikuli Script [9] represents an example of scripting using visual tokens taken from interface screenshots. The result is a script that is primarily textual, with visual tokens.

Modification and Reuse of User Actions

Our inspiration for the advanced features of Tangible Actions comes from very early work in CLIs.

Joy's C shell (1979) was the first system to give sophisticated access to command history [19]. In the C shell, one can access the list of previous commands, as well as re-edit their parameters for command reuse, at a lower interaction cost than re-typing the command. Joy cites Interlisp's REDO command as inspiration [36].

Tangible Actions are able to snap together and thus transfer their output to the next's input. This was inspired by the Unix pipe ("|"), which, according to Ritchie, appeared in Unix in 1972 after being suggested by M. D. McIlroy [30].

Decreasing Interaction Cost

There has been a large amount of work with the motivation of making interaction more efficient, but it has focused on individual actions, rather than programming by example.

In the large display literature, there has been a lot of work on accessing objects that are far away. Khan et al. provide a technique to access portions of the display that are far away with Frisbee [19]. This is similar to our Move Tangible Action, except that the Frisbee effectively acts as a two-way portal. Cao et al.'s work on handheld projectors demonstrates another "portal", where objects can be passed between users [10]. Baudisch et al.'s Drag-and-Pop [5] brings relevant targets closer automatically when the user selects and holds an object, whereas Bezerianos et. al.'s The Vacuum [7] has an explicit interaction technique for bringing objects closer from a specific area.

With Tangible Actions, the user can pause in the middle of a complex multi-step direct manipulation procedure. This idea was explored in Kobayashi and Igarashi's Boomerang [21], where drag-and-drop interactions may be suspended with a throw and catch metaphor, so that the user can perform other necessary actions "in the middle" of the manipulation. Thrown objects are temporarily presented as spinning objects, as if they are mid-air, and may be accessed again when the user is ready.

IMPLEMENTATION AND FEATURES

We implemented three prototypes showcasing Tangible Actions on a Microsoft Surface: a photo-browsing application, an email application and a furniture layout application (Figure 2). They use a common multitouch gesture set.

Actions and Gestures

The workspace of the interface contains objects (either photos, emails or furniture). Users can perform conventional multitouch gestures, such as translation, rotation and scaling. In all applications, a toolbar exists at the top to colour or star objects. In the furniture application, there is another toolbar on the left side of the interface that user can instantiate furniture objects from (left side of Figure 2). A Tangible Action can be created for any action that can be done to an object.

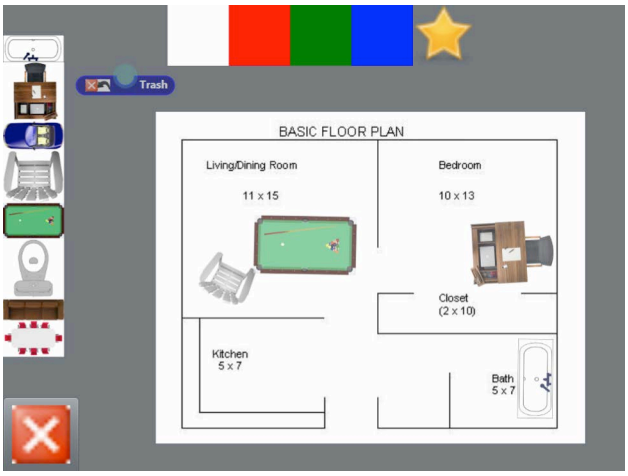

Figure 2. The furniture application. The user places furniture (from left toolbar) on the floor plan (centre). Tangible Actions for colouring and starring can be dragged off the toolbar at the top.

Flexibility of Tangible Action Creation

There are 3 possible ways to create a Tangible Action:

1. Perform a gesture on an object, and activate the Tangible Action from history.
2. Perform an "abstracted" gesture on the background of the interface.
3. Pull a Tangible Action from the toolbar at the top.

In (1), the system initially creates, but hides and disables the Action. In an early prototype, every gesture the user made on an object created a "previewing" Tangible Action, which would fade if not tapped after a few seconds. Even with fading, this caused a massive amount of distraction and interface clutter. Hidden Tangible Actions can be retrieved by as described in the History View section.

In (2), the user is explicitly calling the Tangible Action into existence, as opposed to (1). Thus, we make the Tangible Action "active" right away. We refer to the gesture that the user performs as "abstracted" because they are performing it on an imaginary object, in preparation for future use.

In (3), we have a toolbar because there are some actions that have no obvious gesture mapping. These actions still have value for reuse, so they should be made available as Tangible Actions. The toolbar in our system has a few "colour" actions, as well as a "star" action. The toolbar is visible at the top of Figure 2.

Tangible Action Anatomy

See Figure 5 for a close-up of two Tangible Actions snapped together. Our visual design of Tangible Actions takes inspiration from Scratch, another visual programming language [24]. Main features are the buttons on the left side, the top and bottom notches, and the name and optional parameter of the action on the right side.

The leftmost button is to close or hide the Tangible Action. The action will still be retrievable from the History View.

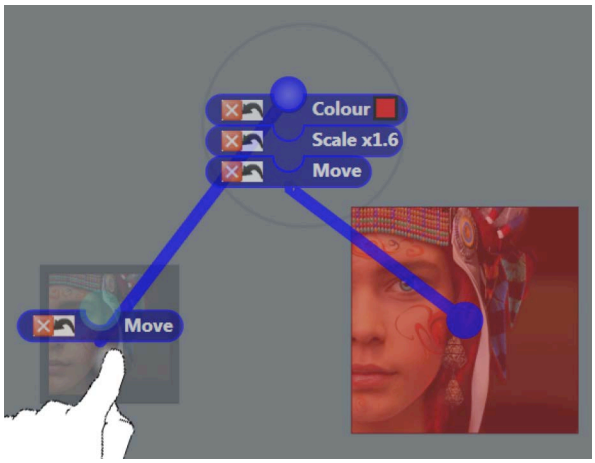

Figure 3. A script of Tangible Actions snapped together: Move, Colour Red, Scale and Move. The resultant object is shown as a preview.

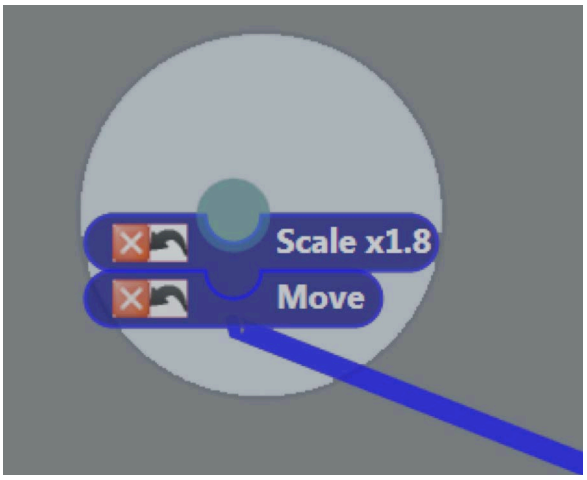

Figure 5. A Scale Tangible Action and a Move Tangible Action snapped together. Visual features of each are described in the Tangible Action Anatomy section.

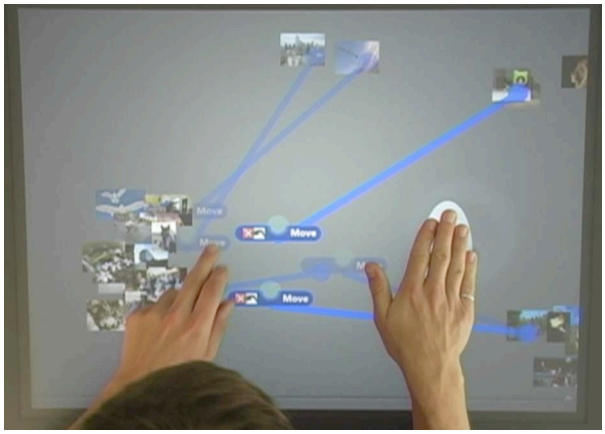

Figure 4. The user places a flat hand down to invoke History View. Here, the user is placing his right hand down. The last few actions performed are shown and may be re-activated.

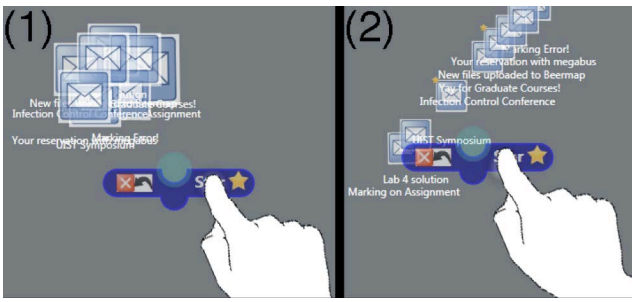

Figure 6. Ad-hoc pile spreading as a Tangible Action approaches a cluster of objects. Objects may be densely packed on the interface (1). When a held Tangible Action approaches them, they spread out so that the Tangible Action can act on them individually (2).

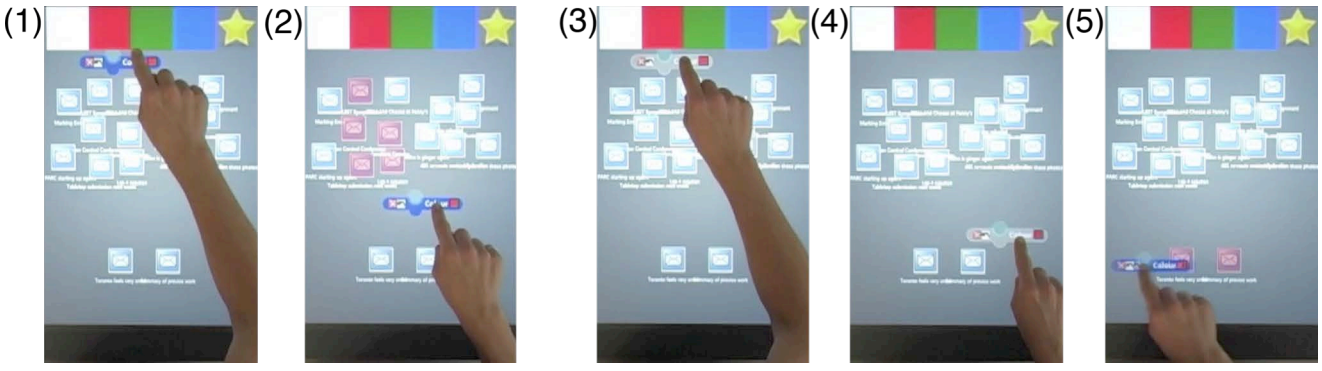

Figure 7. (1,2) Shows a user trying to drag a "Red" Tangible Action off the toolbar to colour the two icons near the bottom of the screen. Since Tangible Actions are "always on", they unintentionally colour several distractor icons. Instead, the user can put a Tangible Action into Inactive Drag Mode by double-tapping, which greys-out the Tangible Action, as shown in (3). The user can safely drag the Tangible Action over the distractor icons (3,4). The user briefly lifts their finger off to return the Tangible Action to the default mode (5).

The second leftmost button is the undo button. Each Tangible Action has a stack of action performances, where it has acted on an object. Tapping on the undo button undoes actions from this stack, regardless of what other Tangible Actions have done on the interface. This is a simplified version of Edward et. al.'s multi-level undo [13].

The notches are directly inspired by Scratch's design language [24]. The inward notch at the top indicates that the Tangible Action can accept input, and is the location of the Tangible Action's "activation point." Initially this is empty, and so the Tangible Action will apply to objects that this notch intersects with on the interface. If another Tangible Action is snapped to this notch (creating a 2-Tangible Action script), it takes input from the previous Tangible Action. When the inward notch is empty, as it is for the Scale action in Figure 5, a light blue circle is shown to indicate the action is accepting input. An outward notch at the top exists for all Tangible Actions except the Move action, which moves objects elsewhere, and the Trash action, which is a penultimate action.

The name of the type of the Tangible Action appears to the right of the middle. To the right of that, if meaningful, there will be the parameter of the Tangible Action, such as the scale factor, rotate angle, or colour to be applied.

Since the Scale Tangible Action accepts two-finger input to change its scale factor (currently 1.8 in Figure 5), it needs a large contact area for the user to gesture on. This appears as the large, light circle in Figure 5. We refer to this as the "gesturable area". See the Tangible Action Modification section for more details.

Flexibility of Tangible Action Invocation

We wanted to allow flexibility of use, so that a Tangible Action can be applied to objects in bulk, but also to individual objects with more hesitance and consideration. Fundamentally, a Tangible Action acts on an object whenever the object is within a small radius of the Tangible Action's activation point. This generates two emergent use cases:

Tossing objects to the Tangible Action to apply the Tangible Action to individual objects.

Dragging the Tangible Action over a set of objects for bulk application. The Tangible Action will automatically be applied to every object that is hit.

Tangible Action Modification

The user primarily modifies Tangible Actions by dragging to move their activation point, but this only modifies which objects the Tangible Action may act on. Parameters of the Tangible Actions can also be modified. For example, the user can modify the end point of the Move action by dragging it. The user can modify the scale factor and the rotation angle of the Scale and Rotate actions, respectively, by placing two fingers on the Tangible action, and manipulating as if the Tangible Action was an object itself. As it is awkward to place and move two fingers on the small area of a Tangible Action, we attach a large "gesturable area" to Tangible Actions that accept two-finger input (Figure 5).

Previewing

Previewing allows users to test the results of an action on an object before committing to it. We show the preview of the action of a Tangible Action on an object whenever the user holds the object over the activation point of a Tangible Action. The appearance of the held object changes to a "wireframe" appearance, and the exact result is showed at the end point (see Figure 3). The user may commit to the preview by releasing their hold over the object. The user can cancel the preview by moving the object away before releasing their hold.

Multiple Tangible Actions and Scripting

We allow multiple Tangible Actions to be combined together into scripts by snapping. When combined into a Tangible Action script, the script will act on objects as if a single Tangible Action. Multiple Tangible Actions that are snapped together operate similar to the Unix pipe [30], where the result of one action feeds into the next. A simple example is shown in Figure 5, with a more complex example in Figure 3.

If a user performs a more complex gestural action, it may be parsed into multiple atomic actions by the system. For example, one sequence of movement by two fingers could be parsed to into a set of Tangible Actions containing a Move, Rotate and Scale.

History View

To view the history of all actions that have been performed on the interface, the user places a flat hand down on the interface (Figure 4). The last few actions performed are shown. A horizontal slider appears on the bottom of the screen and the user may slide this left and right to view the entire history, showing a few actions at a time.

"Actions" in the history are represented as Tangible Actions in preview mode. To activate a Tangible Action for use, the user must tap the previewing Tangible Action. We create a hidden Tangible Action every single time an action is applied to an object, either directly by gesture or indirectly with a Tangible Action. Thus, if a Scale Tangible Action is dragged across 5 objects that are spaced out across the interface, the user will see 5 hidden Tangible Actions in the history view.

Dealing with Clusters of Objects

Our interface does not have an explicit notion of piles, as Bumptop [1] does. However, tightly packed piles occur frequently in our interface, especially when a used has moved several objects to the same location with a Move Tangible Action. As Tangible Actions act indiscriminately on nearby objects, it will always apply to every object in a pile, which is not necessarily what the user wanted. It is inconvenient and inefficient to move the objects around to access only those desired.

Ad-Hoc Pile Spreading

Ad-Hoc Pile Spreading temporarily makes every object in a pile accessible individually for the user. When a held Tangible Action approaches a tight pile of objects, the pile spreads out perpendicular to the Tangible Action (see Fig-

ure 6). As the user moves the Tangible Action left and right, the spacing of the objects changes so the object directly in front of the Tangible Action is easier to access. When the user withdraws or releases the Tangible Action, the objects in the pile return to their original position and size. This is best seen in the video.

We detect "piles" by clustering of all objects on the interface based on a distance threshold. Once clusters are identified, they are spread if the Tangible Action is sufficiently close to at least one of their objects.

Inactive Drag Mode

Inactive Drag Mode is designed to help users apply Tangible Actions to a small subset of objects in a dense field of distractor objects.

In Inactive Drag Mode, when the activation point of the Tangible Action is dragged over objects, it does not automatically apply its action. Rather, it shows a preview, identical to if the object was held over the Tangible Action's activation point. The activation point may now be safely dragged around without unintentionally invoking the action.

The user can put a Tangible Action into Inactive Drag Mode by double-tapping on it. The user may now put a finger down on the activation point and move it around. To apply the Tangible Action's action to individual objects, the user may simply tap another finger nearby. We find this is best done with the index and middle finger. Lifting up the finger on the activation point ends Inactive Drag Mode (see Figure 7).

INFORMAL EVALUATION

We ran an informal evaluation of our system to:

- Determine if Tangible Actions is easy to comprehend
- See when and how participants used Tangible Actions
- Find issues with our implementation

Tangible Actions in its full effectiveness is obviously an expert system. In the space of time available during the evaluation, we would not be able to get users to use all features fully, but we could get them to use most of the novice features of Tangible Actions. This still generated useful and interesting, if critical, observations.

Participants

We recruited participants that would have an affinity for expert systems. These were 4 undergraduate computer science students, between the ages of 18 and 22. All of them owned multitouch phones.

Procedure

The experimenter would spend 4 minutes demonstrating all the features of Tangible Actions. Then, the participants would play with Tangible Actions for about 8 minutes, with the experimenter clarifying functionality if necessary. Then, the participant would perform 4-6 tasks with each of the three applications (photos, email, furniture). The tasks were designed so that the participant could take advantage of repeated actions and scripting, though they were not constrained to doing so. For each application, the participant was told they could do the tasks out of order, as long as they finished all of them. Tasks were instructions that were well-suited to Tangible Actions, such as "Put 6 chairs in the kitchen." and "Colour all chairs in the house red." The two example instructions in the previous sentence were designed so that participants could possibly create a script doing both at simultaneously.

Questionnaire Results

At the end of the study, users filled out a questionnaire with Likert ratings and general comments. Overall, Tangible Actions was found to be easy to learn (4.5/5). However, participants were unsure about whether they liked them (3.125/5) or whether they enabled them to do things better (3/5). Overall, participants felt that Tangible Actions got in the way, responding (2.25/5) for "Tangible Actions did not get in the way". While this is discouraging for this implementation, we did get a number of useful observations. Given the obvious complexity of the system, having all participants agree it was easy to learn is encouraging.

One problem with the Microsoft Surface apparatus is that it has trouble tracking fast-moving contacts. The uninitiated user will not anticipate this, so often they would lose capture of an object while moving it quickly across the interface, and inadvertently create and activate a Move Tangible Action. This led to lots of frustration, with 3/4 participants complaining about it. This is likely the main cause of the low rating for whether Tangible Actions "got in the way".

Observations

The largest issue is that it is hard to convince users that they can use Tangible Actions in a specific context. Users do not perceive redundancies in their own actions as they are performing them. This is a general problem in programming by example systems, not unique to our system.

Placing a hand down on the interface to invoke history view was chosen so that the view could be "triggered anywhere", instead of a button, which is necessarily constrained to a single location on the interface. However, we found that users accidentally triggered it by leaning on the interface, and in one case, with a loose sleeve. It seems that a button in a fixed position on the interface may be a better choice. An alternative is that a specific contact pose could be used [14].

As many of the tasks required participants to apply an action to a subset of all the objects spread across the interface, Inactive Drag Mode was one of the most popular features. However, all participants found it was mechanically difficult to hold on to the Action with one finger and tap with the other.

Most participants enjoyed ad-hoc pile spreading. While there is hysteresis in when the pile spreads and returns to its original position, there is not when the Tangible Action is moving left to right to browse through the objects. This would have been useful, since participants had trouble crossing a Tangible Action over a particular object.

In our implementation, dragging an action over an object applies the action to the object. Dragging an object over an action only shows a preview of the action applied to the object. Participant 3 wished to be able to have the action applied to the object if he dragged the object over the action. Perhaps the invocation of the preview feature should be changed.

Unsurprisingly, clutter was a major problem. Often, participants would leave Tangible Actions on the interface when they were done with them. These would pile up, consuming more and more screen real estate, until they had to be dealt with in one batch, either by deleting them or moving them out of the way. Participant 3 said on his questionnaire: "Screen real-estate is heavily consumed by Tangible Actions". Even though we included a "close all Tangible Actions" button, users would spend a lot of time going around and closing Tangible Actions individually. While this seems to imply that automatically culling Tangible Actions that likely will not be used is a good idea, it is incredibly difficult to determine which Tangible Actions the user wants to use in the future. This again brings us back to the problem of inferring intent.

Conclusions from Evaluation

We argue that Tangible Actions reduces the physical effort of performing repeated actions. However, determining the suitable use of Tangible Actions for a given task requires more cognitive effort than simply performing the task. Overall, users were very pragmatic and did not use advanced features of Tangible Actions, such as snapping them together to create scripts. For example, when they had to rotate several photos by 90 degrees in the photo task, only 2 out of the 4 users took advantage of being able to create a Rotate task. Unfortunately, our evaluation has not determined if Tangible Actions are useful for experts who have become acquainted with the system over time.

DISCUSSION

At the outset, the authors sought to improve the efficiency of multitouch interaction independent of the application. We took the unique approach of turning every instance of user manipulation input into a possible component of a visual programming language, in the same space as the interaction itself. Perhaps unsurprisingly, this led to a few problems with visual and interaction clutter. We will discuss a few of our thoughts on the design of Tangible Actions, and then present some of the unsolved problems that remain.

Limitations

Tangible Actions is not a full programming language, according to Myers' definition of a *programmable* system [27]. Tangible Actions include variables at the moment of their operation, but they are not held on for later use. There is no implementation at all of conditionals or iteration. We initially left this functionality out because of the visual clutter they would create, and the difficulty of visual design. However, these features may not be necessary for the "Just-In-Time" programming tasks we are trying to foster.

Creating a full programming language out of Tangible Actions is likely not be feasible. As operations share the same space as the data it acts upon, abstraction can occur only to a limited degree.

Abstraction of Input

In parsing a user's interaction, there are various levels of abstraction that could be used.

- Literal
- Atomic begin-end manipulative actions
- Inferring the task itself.

We abstracted the literal movements of the user to manipulative actions. The usefulness of literal replaying of multitouch contact movements is suspect, and not very generalizable. Attempting to infer the task itself also seems relatively difficult and will not scale well to new applications.

Comparison to Selection

In the modern GUI, there is a pre-existing method to apply one action to multiple objects: one *selects* the desired objects, and then performs the action. This works well for both manipulative and linguistic interaction. However, this may causes problems when the user is uncertain or there are multiple users.

If the user is certain about the action to be applied yet uncertain about the objects to apply it to, then group selection has issues. For example, if the user has a large collection of photos need colour correction for a subset of them. Using group selection, the group must be decided on before performing the action; selecting the objects to apply the action to is a strict precursor of actually performing the action. The user can either examine each object and add them to the group selection one-by-one, in preparation to apply the action, or apply the action to each object individually. If the cost of performing the interaction is high, such as if it is deep in a hierarchical menu, then either method is time-consuming, and it does not allow the user to be flexible if they are uncertain.

With Tangible Actions, the user can apply the action to one object by invoking it any way they choose, and then drag that action over other objects. The user can easily undo the Tangible Action's effect on the object by pressing its undo button. If the user sees a large clump of objects which definitely need the action applied to them, the user can simply drag the action across the clump quickly, which would be no slower than applying an action to a group of selected objects. However, if the user is certain of the group of objects they want to apply the action to in advance, then group selection will likely be less time-consuming than Tangible Actions.

There are few multi-user, multitouch interfaces that can tell users apart with certainty. The Microsoft Surface [26] and Smart Table [33] do not, but the MERL DiamondTouch [25] does. If users cannot be distinguished, then they must "use" the same set of selected objects. There is no existing implementation of group selection for multiple users on interfaces that are not able to tell the users apart.

Using Tangible Actions may also preserve the direct manipulation metaphor better than group selection. Consider an example of group selection, where the user selects a group of objects and then applies a scale gesture over only one of them. The result will be that all of the objects will be enlarged. It could be argued that this violates the "feel" of directness more than dragging a Scale Tangible Action over each object does.

Properties of Tangible Actions for Multiple Users

We briefly explored the value of the Tangible Actions for multiple users. We came up with four possible reasons for users to share Tangible Actions: authority, skill, physical reach and learning.

For *authority*, one user (the "granter") may want to grant another user (the "grantee") temporary authority to perform an action they would not be able to do directly. The granter is effectively giving the grantee a temporary wrapper for the interface. For example, a private folder owned by the granter. Any other user may not move objects into that folder. The granter can create a Move Action into that folder, and adjust the activation point so that it is reachable by the grantee. The grantee can now move objects into the private folder using the Move Action, and the granter can keep track of what ways objects can end up in the folder.

For *skill*, some novice users may not be physically skilled enough, or suffer from a physical disability that does not allow them to perform a complex, expressive multitouch gestures. Ideally, any interface that is expressive for able-bodied users is also accessible for disabled users. However, an able-bodied user creating Tangible Actions for reuse by a disabled user is a compelling teamwork activity.

For *physical reach*, some areas may not be reachable on large display. There is a large amount of previous work on facilitating reach on large displays reviewed in the Related Work section, but all the techniques discussed are for single users. With Tangible Actions, perhaps multiple users could work collaboratively on one manipulation that spans a larger space than a normal user could reach.

For *learning*, an expert user could teach a novice user how to perform a particular action or they could give them a useful sequence of actions. Experts could share useful, commonly used sequences of Tangible Actions, similar to how snippets of code or instructions for using websites are shared.

Further Applications

In addition to the three applications developed to demonstrate Tangible Actions in this work, they could be useful in a wide variety of applications. The discussion of the potential use of Tangible Actions in multi-user scenarios implies that Tangible Actions could be user to manage files with different rights on a shared interface. Tangible Actions could also be used in a gaming scenario, with one or more users, where a complex gesture triggers a complex spell, and the user want may to recall the spell, or give it to another player to use.

CONCLUSION

We have presented the novel concept of Tangible Actions as an attempt to allow expert users to be more efficient while using manipulative interfaces. Tangible Actions are *continually-created interactive in-situ representations of the user's interface actions that can be modified and reused at a lower interaction cost than re-performing the original interactions.* The concept of Tangible Actions builds on a large body of previous work.

We created a prototype of Tangible Actions implemented in three applications. We found that interaction clutter and accidental invocation is a major problem in our system. However, there is enough potential in the idea of Tangible Actions that we believe this merits further study and development.

FUTURE WORK

While Tangible Actions represent a step up from the tedium of direct manipulation, they still represent a long way from complex automation. It would be valuable to bridge the gap further between Tangible Actions and more powerful scripts. For example, at the moment Tangible Actions apply to all objects indiscriminately; it would be nice to have some filtering or triggers. However, the inclusion of these more complex features may make the ad-hoc use of Tangible Actions more difficult.

Many of the people we have discussed Tangible Actions with have mentioned the use of recording, replaying and modifying actions for interactive art. A good example of art that uses recorded and repeated motion is Bartneck et al's work on Interactive Visual Canons [3]. Another example is in Habib et al.'s SandCanvas [16], where gestures in virtual sand are recorded and replayed to decrease artists' work.

ACKNOWLEDGMENTS

We thank various people for being participants in early prototype testing, especially the members of the Dynamic Graphics Project at the University of Toronto and Wyatt Freeman.

REFERENCES

1. Agarawala, A. and Balakrishnan, B. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. ACM CHI 2006, pp. 1283-1292.
2. Beaudouin-Lafon, M. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. ACM CHI 2000, pp. 446-453.
3. Bartneck, C., & Funk, M. Dancing With Myself: The Interactive Visual Canon Platform. ACM CHI 2009, pp. 3501-3502.
4. Baudisch, P., Tan, D., Collomb, M., Robbins, D., Hinckley, K., Agrawala, M., Zhao, S., and Ramos, G. Phosphor: Explaining Transitions in the User Interface Using Afterglow Effects. ACM UIST 2006, pp. 169-178.
5. Baudisch, P., Cutrell, E., Robbins, D., Czerwinski, M., Tandler, P. Bederson, B., and Zierlinger, A. Drag-and-

Pop and Drag-and-Pick: Techniques for Accessing Remote Screen Content on Touch- and Pen-operated Systems. INTERACT 2003, pp. 57-64.

6. Bezerianos, A., Dragicevic, P., Balakrishnan, R. Mnemonic rendering: An image-based approach for exposing hidden changes in dynamic displays. ACM UIST 2006, pp. 159-168.
7. Bezerianos, A. and Balakrishnan, R. The Vacuum: Facilitating the manipulation of distant objects. ACM CHI 2005, pp. 361-370.
8. Bødker, S. and Andersen, P.B. Complex Mediation. ACM *Human-Computer Interaction*, 20 (4), 2005, pp. 353-402.
9. Chang, T-H., Yeh, T. and Miller, R. GUI Testing Using Computer Vision. ACM CHI 2010, pp 1535-1544.
10. Xiang Cao, Clifton Forlines, and Ravin Balakrishnan. Multi-user interaction using handheld projectors. ACM UIST 2007. pp, 43-52.
11. Clark, H. (1996). Using Language. Cambridge: Cambridge University Press.
12. Cypher, A. (Ed.). (1993). Watch What I Do: Programming by Demonstration. Cambridge, MA: MIT Press.
13. Edwards, K.W., Igarashi, T., LaMarca, A., and Mynatt, E.D. 2000. A temporal model for multi-level undo and redo. ACM UIST 2000. pp 31-40.
14. Freeman, D., Benko, H., Ringel-Morris, M. and Wigdor, D. ShadowGuides: Visualizations for In-Situ Learning of Multitouch and Whole-Hand Gestures. ACM ITS 2009, pp. 165-172.
15. Frohlich, D. (1993) The History and Future of Direct Manipulation. *Behaviour & Information Technology* 12, 6 (1992), pp 315-329.
16. Kazi, R.H., Chua, K.C., Zhao, S., Davis, R., and Low, K. SandCanvas: a multi-touch art medium inspired by sand animation. ACM CHI 2011. pp. 1283-1292.
17. Hutchins, E., Hollan, J. and Norman, D. (1985). Direct Manipulation Interfaces. in Norman, D. and Draper, Stephen W. (Editors), *User Centered System Design: New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, pp. 87-124.
18. Igarashi, T., Matsuoka, S., Kawachiya, S. and Tanaka, H. Interactive beautification: a technique for rapid geometric design. ACM SIGGRAPH 2007.
19. Joy, W. An Introduction to the C Shell. In *UNIX Programmer's Manual, Seventh Edition, Third Berkeley UNIX Distribution,* Dept. of EE & CS, University of California, Berkeley, 1979.
20. Khan, A., Fitzmaurice, G., Almeida, D., Burtnyk, N. and Kurtenback, G. A remote control interface for large displays. ACM UIST 2004, pp 127-136.
21. Kobayashi, M. and Igarashi, T. Boomerang: suspendable drag-and-drop interactions based on a throw-and-catch metaphor. ACM UIST 2007, pp 187-190.
22. Kurlander, D. and Feiner, S. A History-Based Macro By Example System. ACM UIST 1992, pp 99-106.
23. Lieberman, H. (ed.) (2001). Your Wish is My Command: Programming by Example. San Francisco: Morgan Kaufmann.
24. Malan, D. and Leitner, H. Scratch for budding computer scientists. ACM SIGCSE 2007, pp. 223-227.
25. MERL DiamondTouch. http://www.merl.com/projects/DiamondTouch/
26. Microsoft Surface. http://www.microsoft.com/surface
27. Myers, B. Demonstrational Interfaces: A Step Beyond Direct Manipulation. *IEEE Computer* 25, 8 (August 1992), 61-73.
28. Natural User Interface Group. http://nuigroup.com
29. Potter, R. "Just-in-Time Programming". In A. Cypher, editor, Watch What I do: Programming by Demonstration, MIT Press, London, England, 1993.
30. Ritchie, D. 1979. The Evolution of the Unix Time-Sharing System. In *Proceedings of a Symposium on Language Design and Programming Methodology*, Jeffrey M. Tobias (Ed.). Springer-Verlag, London, UK, pp. 25-36.
31. Ruvini, Jean-David. The Challenges of Implicit Programming by Example. ACM IUI 2004.
32. Shneiderman, Ben (1983). Direct Manipulation: A Step Beyond Programming Languages. *Computer*, Vol. 16, No. 8, Aug. 1983, pp. 57-69.
33. SMART Table. http://smarttech.com/table
34. Stuerzlinger, W., Chapuis, O., Phillips, D., and Roussel, N. User interface façades: towards fully adaptable user interfaces. ACM UIST 2006. pp, 309-318
35. Su, S., Paris, S., Aliaga, F., Scull, C., Johnson, S. and Durand, F. (2009). Interactive Visual Histories for Vector Graphics. MIT CSAIL Technical Report June 24, 2009.
36. Teitelman, W. and Masinter, L. The Interlisp programming environment, *Computer vol. 14,* no. 4, 1981, pp. 25–34