Phosphor: Explaining Transitions in the User Interface Using Afterglow Effects

Patrick Baudisch, Desney Tan, Maxime Collomb, Dan Robbins,
Ken Hinckley, Maneesh Agrawala, Shengdong Zhao, and Gonzalo Ramos
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
{baudisch, desney, kenh, dcr}@microsoft.com, maneesh@cs.berkeley.edu
collomb@lirmm.fr, {sszhao, bonzo}@dgp.toronto.edu

ABSTRACT
Sometimes users fail to notice a change that just took place on their display. For example, the user may have accidentally deleted an icon or a remote collaborator may have changed settings in a control panel. Animated transitions can help, but they force users to wait for the animation to complete. This can be cumbersome, especially in situations where users did not need an explanation. We propose a different approach. Phosphor objects show the outcome of their transition instantly; at the same time they explain their change in retrospect. Manipulating a phosphor slider, for example, leaves an afterglow that illustrates how the knob moved. The parallelism of instant outcome and explanation supports both types of users. Users who already understood the transition can continue interacting without delay, while those who are inexperienced or may have been distracted can take time to view the effects at their own pace. We present a framework of transition designs for widgets, icons, and objects in drawing programs. We evaluate phosphor objects in two user studies and report significant performance benefits for phosphor objects.

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

General terms: Design, Human Factors.

Keywords: Phosphor, comic animation, cartoon animation, user interfaces, information visualization, diagrams.

INTRODUCTION

Computer users sometimes make mistakes, such as accidentally deleting an icon or filing it into the wrong folder. Similarly, unexpected things may occur in collaboration scenarios. Users trying to replicate a process demonstrated by a collaborator may later realize that they missed some of the steps. This is particularly difficult for actions that leave no trace, such as shortcut commands.

The potential changes that users need to keep track of continues to rise with increasing user interface complexity, more concurrently running applications, large screens where the user may be attending to the wrong location, and the possibility of remote collaboration. Without knowing what changed and how it changed, users can find it hard to detect and correct unintended or unexpected actions.

Animated transitions have been proposed to help users understand changes in the user interface [9, 19] and have found their way into a range of products. *Windows Media Player 10*, for example, hides its play controls in fullscreen mode by slowly moving them off screen. While this can help users understand where the controls went and how to get them back, it also introduces "lag" into the interaction, i.e., it forces users to wait for the animation to complete. For experienced users who do not need an explanation, this forced pause can be cumbersome and may break their concentration.

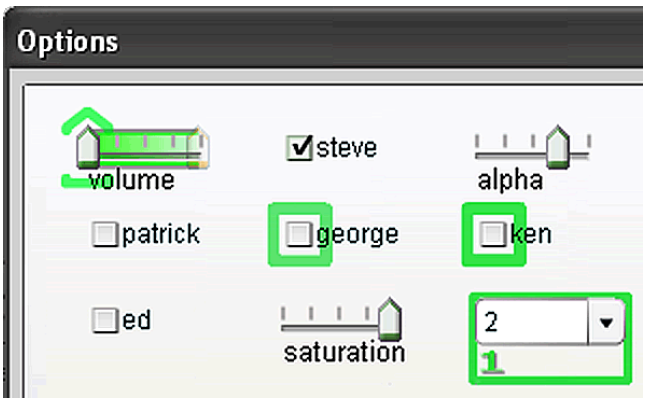

Figure 1: These phosphor widgets use green afterglow effects to show how they have changed. The slider labeled "volume" was dragged all the way to the left. Two of the checkboxes in the next row were unchecked. The combo box was set from 1 to 2.

PHOSPHOR USER INTERFACE OBJECTS

We propose explaining user interface transitions without forcing users to wait. We define a *phosphor transition* as a transition that:

1. shows the outcome of the change *instantly* and
2. explains the change in retrospect using a diagrammatic depiction

The space of retrospective diagrammatic descriptions encompasses a great number of possible designs. In this paper, we concentrate on a specific subset based on the notion of afterglow. Figure 1 shows an example. When a user op-

UIST'06, October 15–18, 2006, Montreux, Switzerland.

erates the shown *phosphor widgets*—using the mouse or a shortcut—a stylized afterglow effect shows how they have changed. In some cases, such as the slider, the afterglow is an almost realistic depiction of the motion that took place during the change. Widgets that change in more complex ways, such as the combo box, are provided with a more abstract and symbolic afterglow in order to limit clutter.

Phosphor widgets are designed to focus users' attention on objects that have changed. If users believe a manipulation took place in error, they can undo the action by returning the widget to the previous state suggested by the afterglow. Afterglow also helps users to observe changes that they have not initiated and hence provides better understanding in scenarios such as remote collaboration.

Afterglow effects are typically set to fade over a period of a few seconds. Actions occurring in rapid succession will therefore result in multiple concurrent afterglow effects. This is intended to help users catch up with fast bursts of activity as might occur during demonstrations or collaborative work.

Resulting benefits

The proposed approach differs substantially from animated transitions: animated transitions explain the transition *and then* continue the regular execution of the program; phosphor transitions do both *at the same time*.

This parallelism results in three main benefits: (1) Users can choose whether to attend to the explanation or to continue with the regular program execution. Users are never forced to wait. (2) Since users are never forced to wait, additional display time comes at a low price. Inexperienced or distracted users can therefore be accommodated with increased afterglow durations. (3) Since display time comes at a low price, application designers can pick a reasonable upper bound. This frees them from having to hand-optimize duration—a major challenge faced by designers of animated transitions.

The use of phosphor widgets introduces a tension between screen real estate and interaction time. Because phosphor widgets are susceptible to clutter, they require careful design.

In the remainder of this paper, we give a brief overview of the related work. Then we take a closer look at the "visual language" of phosphor. We present designs for transitions for different types of interface objects and explain how to minimize clutter. After a brief description of our implementations we present two user studies. The first study finds significant performance benefits for phosphor over a control condition: Participants performed a simulated collaboration task faster when widgets were provided with an afterglow. The second user study finds that phosphor's task performance is similar or better to animated transitions. We conclude with a summary of our findings and an outlook to future work.

RELATED WORK

Two main fields of related work for Phosphor are animated transitions and diagrammatic explanations.

Animated Transitions

Animated transitions are one of the eight classes of animation in the user interface [2]. Benefits of animated transitions include that they can help increase the saliency of notifications [4], draw attention to peripheral displays, such as stock tickers [24], and that they can help illustrate causal relationships [33]. Animated transitions can help users follow transitions between views [3], e.g., in applications displaying complex data, such as trees [28]. By adding effects inspired by cartoons such as anticipation and follow-through, researchers have obtained a more lifelike effect (cartoon animation [9, 31]).

Research has not converged on consistent results regarding the efficiency of animated displays [32]. Animated illustrations may require more cognitive load than static ones [21]. Psychophysics research has shown that most users have difficulty tracking five or more objects [8, 36, 25]. Motion is hard to ignore and may thus cause users to be distracted by animated transitions [4].

Stasko [29] points out that animation duration is a crucial factor in the design of animation. To minimize lag, an animation should be fast; making an animation too fast, however, may lose the user. Researchers exploring animation durations have found that 300ms can work well for simple scrolling transition [19], while comprehending 3D transitions can require several seconds [27]. Optimum animation speed depends on user- and situation-specific factors such as familiarity, expectation, attentiveness, and perceptual abilities and therefore are difficult to predict.

While designers of phosphor objects also need to set the duration for fading the afterglow, the question is less crucial because the afterglow does not prevent users from continuing their task.

Diagrams in information visualization

In the fields of visualization and graphics, researchers have proposed illustrating dynamic phenomena using static depictions. Diagrammatic illustrations are amenable to printing [1], and can help users discover trends in large sets of motion data [10]. On the flip side, users do *not* process diagrams immediately and as a whole; users first have to discover the best order to process the information [7].

Diagrammatic summaries come in many different styles. Feiner borrows principles from technical illustration [11], while Hill and Hollan use them to illustrate the past usage of a document [16]. Carefully selected individual frames can be combined to form a *strobe effect* (*Action Synopsis* [1]). *Chronovolumes* combine successive frames into a continuous *motion blur*. They use color transitions to depict the progression of time [35]. *Speed lines* [22] are a more abstract type of motion blur created using non-photorealistic rendering [26]. Speed lines have also been used to enhance the experience of animation sequences in video games [14] and to help users make sense of game map overviews [17].

Comics use static depictions of dynamic contents that have been adopted by user interface research such as Comic Chat [20]. The individual frames of a comic are multiplexed in space, unlike cartoons that show them in temporal succession [23]. The visual language of phosphor is similar to that of comic books in that both depict the past; the act of running is shown by adding motion blur *behind* the character; a punch is shown as a fist that has already followed through. Story boards in contrast tend to depict the future.

Diagrammatic transitions in the user interface

There are only a few examples of diagrammatic cues depicting motion in user interfaces. Mac OS X complements animation with a motion blur when iconifying windows. *High-density cursor* improves target acquisition by adding a strobe effect to the mouse trajectory [5]. Gutwin and Penner showed that similar cues applied to telepointers can improve collaboration [13]. Kaptelinin et al. showed that a static cue can improve the reading performance of scrolled pages. Their design minimizes clutter by using only an outline to highlight "new" screen content [18]. Our work on phosphor is different in that it focuses on individual objects, rather than view navigation. Our paper generalizes the transition styles of *drag-and-pop*, an interaction technique for accessing distant content on wall-size displays [6].

THE VISUAL LANGUAGE OF PHOSPHOR

In this section, we show how the design space of phosphor is applicable to many objects and transitions, such as icons, windows, or objects in a drawing program. Such objects may experience a broad range of transitions, such as rotation, change in stacking order, or changes in opacity.

Figure 2 illustrates the general concept for creating a phosphor transition. First, we envision an animated transition. Second, we conceptualize this transition as a static depiction by projecting along the time axis. The result is a single image consisting of the *initial state* of the transition, the *final state*, and the *path* in between.

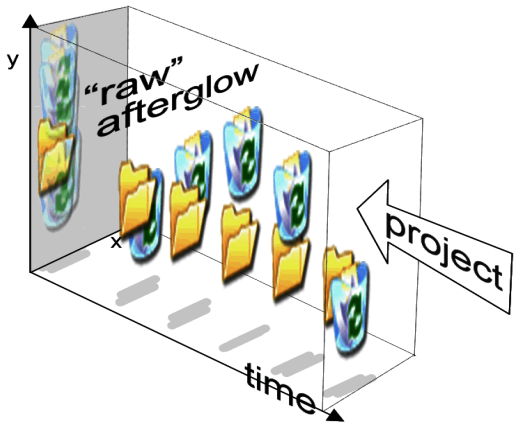

Figure 2: The afterglow of phosphor objects is generated by projecting an animated transition along its time axis.

For translations, e.g., the knob of a slider, this process is fairly straightforward and can be implemented using a motion blur effect. For in-place transitions, such as a rotation or change in opacity, however, all frames of the animation fall onto the same location. We present measures to avoid occlusion in such cases. We also present cues that compensate for the loss of temporal order during the projection step. We first present visual styles for paths.

For the sake of visual consistency, all examples in this section use the same visual objects, here two desktop icons. These are intended as exemplars for phosphor objects in general.

Paths styles: strobe, motion blur, and speed lines

Figure 3 shows the three basic path styles we have used, all inspired by comic books [23].

The *strobe style* shown in Figure 3a consists of a finite number of partially overlapping frames. It can help convey complex transitions by breaking it down into a frame-by-frame illustration.

The *motion blur style* shown in Figure 3b is generated from a single copy of the object by first compressing it to single-pixel-width and then by stretching it to the desired length. This approach leads to less clutter, as the individual frames dissolve into a single gestalt.

In a literal implementation of strobes and motion blur, long paths would be practically invisible, because each of the n frames the path is composed from has opacity of only $1/n$ [5]. We ensure path visibility by increasing path opacity to a point were paths are clearly visible. Path opacity is subject to a tradeoff between visibility and clutter, but we obtained good results with opacities of 50% for the densest part of the path.

For most of our applications, as well as both user studies reported in this paper, we use the *speed lines style* shown in Figure 3c. This style consists of a pair of edge lines and a center line on top of a background surface, as used in [6]. It provides a good indication of orientation and causes less clutter than the motion blur design. By using characteristic colors sampled from the object, speed line paths often also resemble their parent objects more than the washed out colors of the actual motion blur. This helps visually match paths with parent objects and visually separate them from other intersecting paths (see also Figure 11).

Figure 3: Basic path styles (a) strobe (b) motion blur, and (c) speed lines.

We make paths transparent to mouse input to allow users to interact with screen content temporarily occluded by a path. If a transition is repeated while its afterglow is still on screen (for example, when manually "nudging" a geometric

object in a drawing program), we keep the initial state in place and only extend the path.

Reestablishing the time dimension

Paths do not inherently offer any sense of directionality. So, to distinguish an object moving from A to B from an object moving from B to A, we add three cues to reinforce temporal order,

1. Fading "old" path segments: We render path segments increasingly translucent the further away they are from the final state. Accordingly, fading an afterglow causes the path to disappear initial state-first (Figure 4a).

Figure 4: (a) Paths fade initial state first. (b) The temporal order of this strobe-style path is emphasized by stacking frames in chronological order.

2. Stacking newer frames on top of older frames: We always render the path on top of the initial state and the final state on top of everything. This also guarantees that the final state is never occluded and always fully readable. We use the same concept among the individual frames of strobe style paths (Figure 4b).

3. Distinctive initial states: In order to help users distinguish initial and final state, we render them differently if possible. We always render the object's final state as is; only then can we guarantee that it is legible and immediately available for further interaction.. In Figure 4a we rendered the initial state as an outline to suggest the *absence* of that object, as appropriate for a move operation. We discuss other styles in a later section.

We decided against other potentially useful cues for temporal order, such as texturing paths with arrow symbols or the use of animated textures. Animated textures do not impact the readability or accessibility of the object's final state, but they seemed distracting. While tapered paths [17] could be used for this purpose, we instead chose to use this cue to depict transient operations as described below.

Depicting the operation causing the translation

We use different path shapes and initial states to disambiguate whether the translation of an object occurred because the object was moved, copied, or filed in a subfolder

Move vs. copy: A copy operation is a move operation that does not disturb the initial state. We therefore depict copy operations as move operations with a solid initial state (Figure 5b).

Figure 5: Whether a folder is being (a) moved or (b) copied is determined by the initial state visuals.

Concave paths indicate transience: Some translations are transient, such as a translation suggested by the system but not yet confirmed by the user. We depict such translations using paths with a narrow midriff section to suggest that the final state is connected to the initial state using an elastic rubber band that will eventually pull the final state back (Figure 6a). The same effect can also be used to create and remove a temporary copy (Figure 6b) as done by *drag-and-pop* [6]. Since the path is *attached* to the initial state we do not fade it. Note how the diagrammatic nature of phosphor allows depicting the future.

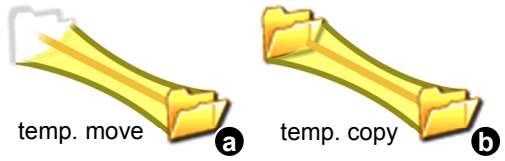

Figure 6: A narrow midriff on this path indicates that this folder was (a) moved or (b) copied *temporarily*.

Tapered paths indicate a child-of relationship: Sometimes an object is spawned or unveiled by another object, such as a child window being spawned by its parent window or a folder being extracted from another folder. To avoid confusion with a copy operation, we use a tapered path, suggesting that an object needs to be shrunk before it can fit into another object (Figure 7a) and that a restored object starts small and ends large (Figure 7b).

Figure 7: The tapered path indicates that (a) the folder at the bottom left was dropped into the other folder or (b) expanded from it.

Transitions that do not involve translation

When collapsing the time dimension for transitions that lack translation, all frames project onto the same location. We alleviate this problem by adding motion to the animated transition before we project it so that phosphor can depict time in screen space. Figure 8 shows an example. (a) Bringing an object to the front can be thought of as a transition exclusively in Z-direction [9]. (b) Before projecting, we add motion to the path such that it evades the other object. (c) Collapsing this second animation provides the desired path.

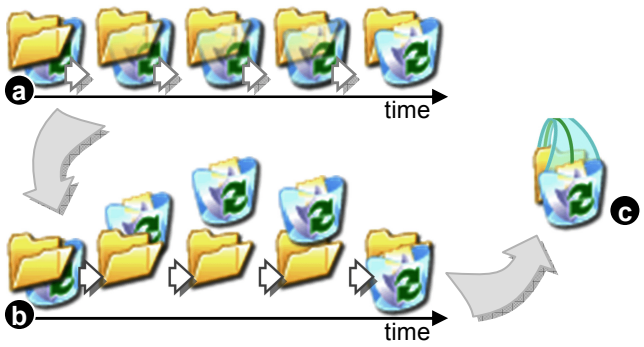

Figure 8: (a) initial animation, (b) adding vertical motion, and (c) resulting phosphor transition.

Revealing the initial state: Adding motion to the path reveals the path, yet the initial state remains occluded. We ensure the visibility of the initial state by translating it forward in time along the path, as shown in Figure 9a. In cases where we can expect users to know that this translation is not part of the transition, we can omit the path (Figure 9b and c, as well as the combo box in Figure 1).

Figure 9: (a) The initial state of this scaled-up object was revealed by moving it out along the path. (b) If unambiguous, we may omit the path. (c) The matching scaling down transition.

Out-of-band properties in the initial state: In many of our examples, we repurposed properties of the initial state. We used texture to convey temporal order (Figure 4), size to convey an inclusion relationship (Figure 7), and location to avoid occlusion (Figure 9). But how can users know that the object in Figure 4 did not change from an outline to a solid object, that nothing shrank in Figure 7, and that nothing moved in Figure 9? We used what we call *out-of-band signals*, another example of which is shown in Figure 10a and b.

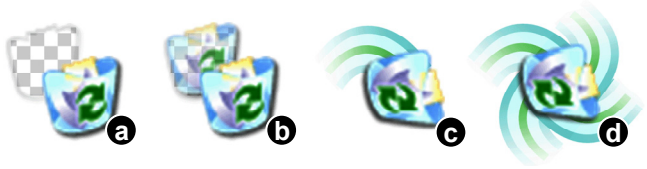

Figure 10: Transition (a) from transparent to opaque and (b) from 50% transparent to opaque. (c) A rotation may be confused with a translation, unless (d) an out-of-band cue is added.

This transition shows an increase in opacity by showing a "transparent" initial state. Using real transparency would cause the initial state to be invisible. We therefore use a symbolic representation instead, here a checkerboard pattern that shines through. This approach is commonly used in painting programs, such as Adobe Photoshop. It works, because checker boards are unlikely to occur in the application area of interest, such as photographs or icons. The limitation of this approach is that the choice of out-of-band signals is application-specific. Application areas that include checker board textures, for example, require using a different stimulus. We have used the out-of-band concept for a wide range of transitions, including rotation (Figure 10c and d).

Transitions that involve multiple objects

If paths cross at an angle, speed line gestalt is sufficient to assure readability (Figure 11a). During pilot testing, we found that up to ten simultaneous translations on the screen were visually separable, even when they where of the same color scheme. For cases of exact overlap (Figure 11b), readability can be maintained by letting paths avoid each other (Figure 11c).

Figure 11: (a) In the general case, path overlap is not a problem. (b) Objects trading places (c) is better handled with paths avoiding each other.

If large numbers of objects move, we may reduce clutter by shortening paths (Figure 12a, see also [30]). We can use a single path for groups of objects performing the same transition (Figure 12b).

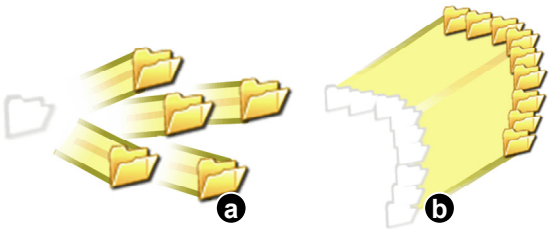

Figure 12: Avoiding clutter by (a) shortening paths and (b) using a single path object.

IMPLEMENTATION

In order to try out our concepts and to run the user studies reported below, we created two prototypical implementations of phosphor.

Figure 13 shows a screenshot from our first prototype. It simulates a computer desktop and allows us to create a variety of different transition types. It is implemented in Delphi using the .NET framework. The prototype allows loading arbitrary graphics for objects and if desired separate graphics for their initial states. It generates different shapes of paths using Bezier curves with four control points. Colors of speed line path are generated automatically by sampling and averaging colors from the respective objects. The prototype allows customizing path opacity; the defaults are 12.5% to 50% opacity for the path fill and 25% to 87.5% for path edges. Finally, this prototype also offers animated transitions.

Our second prototype is shown in Figure 14, a close-up in Figure 1. It is implemented in Macromedia Flash and offers

phosphor versions of standard GUI widgets. Widgets are implemented as *Flash components* [12], which allows reusing them in other Flash applications.

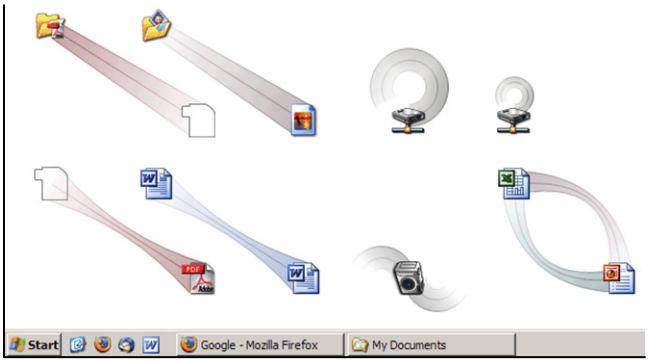

Figure 13: Our Delphi-based prototype showing several phosphor transitions.

To objectively evaluate user performance using phosphor transitions we carried out two user studies.

USER STUDY 1: FOLLOWING A COLLABORATOR

The purpose of the first study was to examine whether phosphor helps users visually track transitions in the user interface. The participant's task was to watch a (simulated) collaborator adjust widgets in a control panel dialog. Then participants had to demonstrate their comprehension by undoing as many of the observed actions as possible. Our main hypothesis was that the phosphor interface condition would lead to better perception and memorization of the activities, which we would measure as faster task performance.

Task

Each trial took place in the following six steps. (1) Users clicked a "go" button. (2) A dialog appeared and the user's mouse was disabled. The dialog contained a mix of sliders, combo boxes, and checkboxes laid out in a regular 5x8 grid as shown in Figure 14. Layout and labeling of widgets changed each trial. To indicate the non-interactivity, the window bar of the dialog was shown in gray. (3) After a pause of 500 milliseconds, the mouse pointer started traversing the screen and consecutively adjusted six of the forty widgets in the grid, i.e., two sliders, two checkboxes, and two combo boxes. Simulating a human motion pattern, the pointer would hover 200ms over a widget before adjusting it. There were three *playback speed* conditions. In playback speed conditions "medium" the mouse traversal from one widget to the next took 600ms, 800ms, or 1000ms in random order, always resulting in the same overall duration of 4800 ms. In the "slow" and "fast" playback speed conditions all traversal and hover times were multiplied by factors 1.3 and 0.7 respectively. (4) There was a 4000ms pause. At the end of that pause, all of the 4000ms afterglow effects had faded completely. (5) The mouse was re-enabled, the window bar of the dialog turned blue, and a text was displayed instructing participants to start undoing as many of the observed changes as possible. (6) The participant adjusted as many widgets as they desired. Then they clicked the dialog's "OK" button, which closed the dialog and completed the trial. Task time was counted from the moment the dialog turned interactive until the moment the user hit the "OK" button.

Interfaces

There were two interface conditions, *phosphor* and *control*. The *phosphor* condition was identical to the control condition, except that widgets displayed an afterglow after being adjusted by script or participant as shown in Figure 1. Each afterglow faded independently after 4000ms.

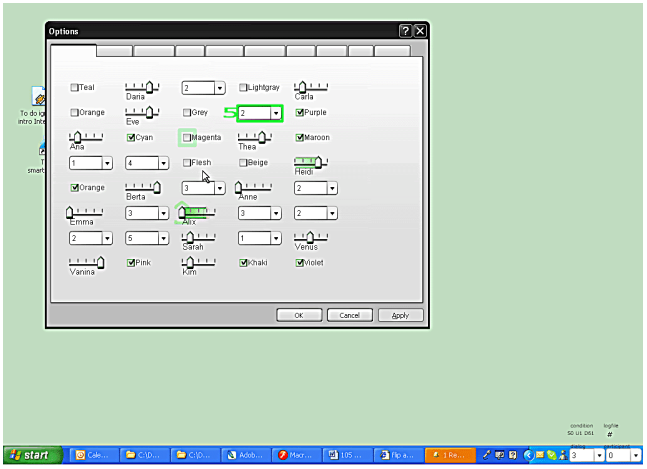

Figure 14: The apparatus used in the first user study. The user's task was to undo as many of the six adjustments observed.

Experimental design

The study design was within subjects 2 × 3 (user interface x playback speed) with 8 repetitions for each cell. For each trial, we recorded completion time and error. Error was any difference between the initial state before the automated traversal and the final state after the user adjustments; clicking "OK" without making any adjustments, for example, resulting in six errors. Interface order and speed factors were counterbalanced. Dialog layout was randomized.

Participants received training upfront and at the beginning of each block. They filled in a questionnaire at the end of the study related to subjective preference, learnability of phosphor transitions, and usefulness. The study took about 20 min per participant.

Apparatus

The experiment was run on a PC running WindowsXP with a 17" LCD monitor, at a resolution of 1280x1024 pixels. The interface used in this study was implemented in Macromedia Flash as described earlier. The optical Dell mouse was set to a medium mouse speed.

Participants

12 university students (1 female) between the ages of 24 and 30 participated in this study. All had experience with graphical user interfaces and mice and had normal or corrected to normal vision and color vision.

Hypotheses

We had two hypotheses: (H1) Participants would be able to perceive and memorize changes better in the phosphor condition. (H2) We expected the increased performance to cause participants to subjectively prefer the phosphor interface. We had no particular hypothesis about the impact of playback speed.

Results

We analyzed the data for this experiment at the summary level, taking the median of the completion times and the mean of the errors over the 8 trials in each condition.. The dependent variables were error rate and task completion time. We analyzed each variable using a 2 (interface) × 3 (playback speed) repeated measures analysis of variance (RM-ANOVA).

For task completion time, we observed a significant main effect of interface ($F(1,11)=20.07$, $p=.001$), with users completing the task faster with the phosphor interface (Med=7.17 s) than with the control interface (Med=9.35 s).

We also observed a significant main effect of playback speed ($F(2,22)=4.06$, $p=.031$). Paired comparisons using Bonferroni corrections showed that this difference was driven by significant differences between the fastest condition (Med=7.34 s) with the slowest condition (Med=9.01 s, $p=.02$). We observed no significant interactions between the two factors.

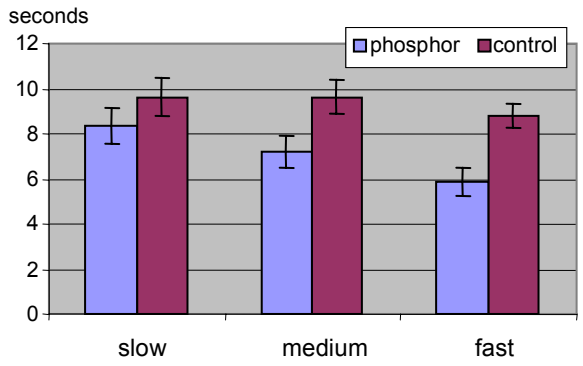

Figure 15: Task completion times of the first user study. Participants performed the undo task faster when widgets were provided with a phosphor effect (times in seconds, +/- standard error of mean).

We observed no significant effects when we examined error rates. Users made on average about 3.1 errors, i.e., they were able to undo about half of the observed changes, with no significant differences across conditions. Hence, while Phosphor seemed to help with the speed at which users were able to retrieve answers from memory, our task and error rate metric was not sensitive enough to pick up differences when using this interface.

At the end of the experiment, participants answered a questionnaire. All 12 participants preferred the phosphor interface to the control interface. On a 7-point Likert scale, participants agreed that the phosphor interface helped them remember more changes (mean value: 5.9). Participants also expressed that it did *not* take long to get used to the phosphor interface (mean value: 2.6).

Discussion

The timing data supports our first hypothesis. We believe that the difference in completion times represented the level of confidence that users had in their responses. When they used the Phosphor interface, they were more certain of the answers and spent less time thinking when undoing changes. This performance difference also manifested itself in participants' subjective preference.

Interestingly, participants performed faster in the faster playback conditions. One possible interpretation is that the reduced time spent watching the actions helped participants keep the actions in their working memory.

USER STUDY 2: COMPARISON WITH ANIMATION

The purpose of the second user study was to compare phosphor with animated transitions. While phosphor effects have the benefit of not introducing lag, we were wondering if this benefit would come at the expense of reduced task performance when compared with animation.

We also used this experiment to learn more about how multiple simultaneous transition effects and distractors impact user performance and user preference.

Tasks

This study simulated the situation of a user who has copied and pasted one or more files into a folder window and who now tries to visually verify whether the expected action has taken place.

Each trial proceeded as follows. (1) A simulated windows desktop screen was displayed. The screen contained 11 small windows and 25 icons of the same file type. (2) For 1200ms, the interface highlighted three icons and one target window as shown in Figure 16a. (3) The highlights were removed to prevent users from completing the task by simply tracking the highlights. There was a 500ms pause. (4) Three icons moved across the screen. (5) Participants pressed the "Y" key if they felt that the shown transition corresponded to the highlighting showed earlier; otherwise they pressed the "N" key. The answer keys could be pressed as soon as the transition *started* and participants were encouraged to press the correct key as soon as they knew the answer.

There were four possible outcomes for each trial, each of which occurred equally often. *Correct:* All three icons successfully reached the target window (Figure 16b). *ErrorNeighbor:* An incorrect icon moved; it was located closely to the expected one. *ErrorOther:* An incorrect icon moved; it was located far away from the expected one. *ErrorUndershoot:* The right icons moved, but one of them did not reach the target folder because it over or undershot.

There were three versions of this task called *singleIcon*, *tripleIcon*, and *distractor*. The *tripleIcon* task was described above and shown in Figure 16. The *singleIcon* task was identical to *tripleIcon*, but only a single icon was highlighted, moved, and needed to be verified. The *distractor*

task was a mix of both conditions. Only a single icon was highlighted and needed to be verified, but an additional two icons moved. The purpose of the distractor task was to provide insight about how motion on the screen affects user performance with animation and phosphor.

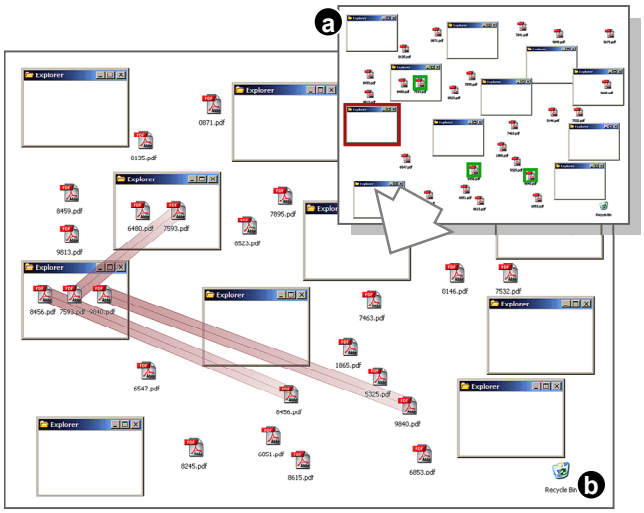

Figure 16: The *tripleIcon* task. (a) Three icons are highlighted in green and a target window in red, then (b) three icons perform a transition, here shown using the *phosphor* interface condition. In this example the transition matches the earlier highlighting, so the correct response is to press "Y".

Interfaces

We tested two interface conditions. The *phosphor interface* (Figure 16) used the speed line design described earlier and the copy visuals shown in Figure 5. Phosphor cues were shown the moment the transition was triggered and were removed when the participant hit the answer button. The *animation interface* used slow-in/slow-out animation at 25 frames per second. If multiple icons moved, they moved in synchrony. There were five animation conditions with animation durations of 125ms, 250ms, 500ms, 1000ms, and 2000ms, covering the combined range from several experiments reported in the related work, e.g. [19, 28].

Experimental design

The study was a 6 × (Interface: phosphor, 5 × animation) × 3 (Task: SingleIcon, TripleIcon, Distractor) × 4 (Outcome: Correct, ErrorUndershoot, ErrorNeighbor, ErrorOther) within-subjects design. During the study, participants performed each task in its entirety using all six interfaces before moving to the next task. Interface and task order were counterbalanced. Participants performed the same 60 trials (15 icon layouts x 4 outcomes) for each of the six interfaces of a given task in different randomized orders. This allowed us to control for the difficulty of the trial. Pilot testing and analysis on the study data showed no learning effects. For each trial, we recorded task completion time and error. Task time was counted from the moment the transitions started until the moment the participant pressed one of the answer keys.

Participants received training at the beginning of the experiment and prior to starting each block. At the end of the study, they filled in a questionnaire. Overall, the study took about 90 min per participant.

Apparatus

The experiment was run on three PCs running WindowsXP with LCD screens driven by NVIDIA graphics cards and offering a 60Hz refresh rate. The test program was 1024x768 large (11"/28cm wide). Participants interacted with the system using a standard PC keyboard.

Participants

Twelve volunteers (4 female) between the ages of 22 and 35 participated in this study. All had experience with graphical user interfaces and had normal or corrected to normal vision and color vision. Each received a small gratuity for their time.

Hypotheses

Our main hypothesis was that the phosphor interface, which allows users to view transitions at their own pace, would perform as well as the best animation condition. We also assumed that the harder tasks would impact task performance across interface conditions, but we had no clear hypothesis about how.

Results

We analyzed the performance data at the summary level, averaging the 15 trials within each condition. We used a 6 (Interface) × 3 (Task) × 4 (Outcome) repeated measures analysis of variance (RM-ANOVA) for each of the dependent measures, trial time and error rate.

Task time phosphor vs. animation: For average trial time, we observed a main effect of Interface ($F(5,55)=243.596$, $p<.001$). Pairwise comparisons with Bonferroni correction showed significance across all pairs of interfaces, except for the Phosphor × 125ms, the 125ms × 250ms, and the 250ms × 500ms conditions (Figure 17a and c).

Task time across tasks: We observed a significant main effect of Task ($F(2,22)=8.974$, $p<.001$). In pairwise comparisons using Bonferroni correction, we found that users were significantly slower in the TripleIcon condition (M=1160.4s) as compared to both the Distractor condition (M=995.9, $p<.020$) and the SingleIcon condition (M=1062.0, $p<.019$). As one would expect, tracking three icons takes longer that tracking a single icon.

A planned contrast on the Phosphor task times found that all three task conditions were significantly different from each other ($F_{2,22} = 17.68$, $p < .001$). Trials of the TripleIcon task (1010.72ms) took participants 56% longer than the SingleIcon task (645.54ms). The additional two distractor icons of the distractor task increased task time by 25% to 804.25ms per trial for the distractor task.

Task time across outcomes: Finally, we found a significant main effect of Outcome ($F(3,33)=966.709$, $p<.001$). Pairwise comparisons showed all Outcomes significantly different from each other. The ErrorNeighbor and ErrorOther

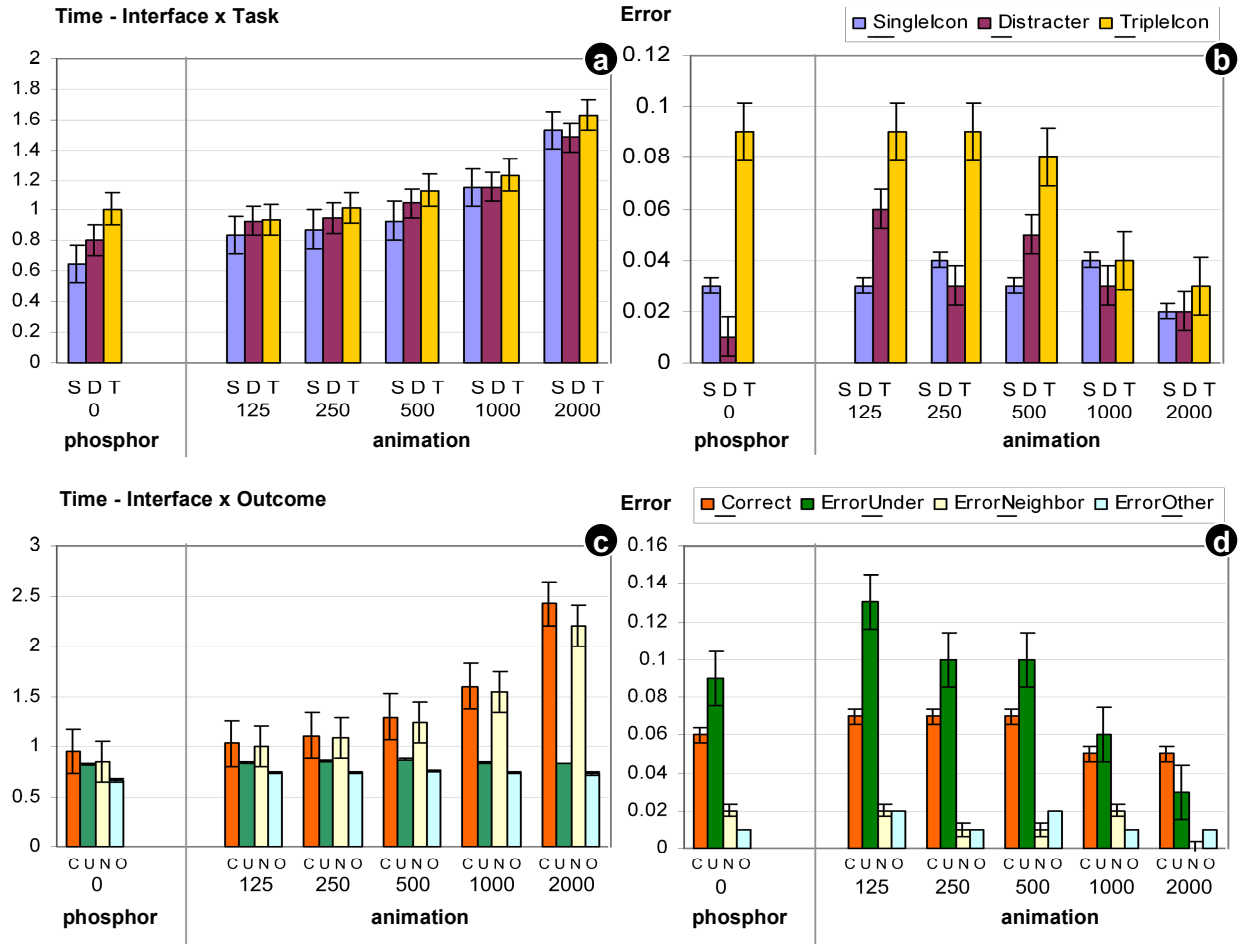

Figure 17: (a) Task time in seconds and (b) error rate by interface and task. (c) Task time in seconds and (d) error rate by interface and outcome (+/- standard error of the mean).

outcomes were faster than the other two outcomes (Figure 17c). These two error conditions involved a wrong icon moving, which participants detected early on and dismissed these trials instantly. Correct and ErrorUndershoot outcomes, on the other hand, required participants to wait for the animated transitions to complete. Consequently, we saw higher task times overall and in particular a strong negative impact in the slower animation conditions.

Error rate: Similarly, for the error rate metric, we observed a main effect of Interface ($F(5,55)=4.704$, $p<.001$), driven mainly by differences between the 2000ms animation Interface and the Phosphor, 125ms, 250ms, and 500ms conditions. We also saw a main effect of Task ($F(2,22)=16.985$, $p<.001$). As expected errors in the TripleIcon condition were significantly higher than that of Distractor ($p<.006$) and SingleIcon ($p<.001$) tasks (Figure 17d). Finally we saw a main effect of Outcome ($F(3,33)=21.243$, $p<.001$), with all pairs significantly different except for the correct v. ErrorUndershoot and ErrorNeighbor vs. ErrorOther. ErrorOther and ErrorUndershoot were easier to detect and therefore led to low error rates (Figure 17b). The error in the ErrorNeighbor task, on the other hand, was easily missed and led to an error rate even higher than the Correct outcome.

Subjective preference: Participants selected one "favorite" interface condition after completing each task. To simplify selection, we offered the four mnemonic choices 'super fast', 'fast', 'slower', 'super slow', and Phosphor. Preferences varied between tasks. For the *singleIcon* and the *Distractor* tasks, the fast condition was a favorite (7 out of 11 participants). For the *tripleIcon* tasks, preferences were more balanced with 3 participants each preferring the fast, slow, and phosphor. The reason most commonly given for the lower popularity of the phosphor interface was that it was more distracting, especially in the distractor task.

The slow animation conditions received very low satisfaction rating throughout, despite their superior error rate. Several participants expressed a strong dislike for the wait time caused by these conditions.

Discussion

The findings of our second user study indicate that the performance of the phosphor interface is comparable to animated transitions. For all pairs but the fastest animation condition, Phosphor was significantly faster. For the 125 ms animation condition, there was no significant difference between conditions. This suggests that the benefits of phosphor do *not* come at the expense of reduced task performance.

At the same time, the second study suggests there may be limits to the space-for-time approach behind phosphor. With paths reaching across the entire screen, users cannot foveate the entire path at once, and the resulting display can become distracting, as the subjective preference data indicates. While we may be able to push this limit out by fine tuning the path visuals (e.g., thinner paths with lower opacity), the contrast with the strong positive findings of the first study indicates that the greatest benefits of phosphor might lie in the space of *localized* effects (see also *Proximity Compatibility Principle* [34]).

CONCLUSIONS

Phosphor is a technique for explaining transitions in the user interface. Unlike animated transitions, it never forces users to wait. Our first study indicates that phosphor transitions help improve users' ability to process changes in the user interface. Our second study indicates that the benefits of phosphor over animated transitions do not come at the expense of task performance.

As future work we plan to continue to investigate the learnability of our design. We also plan to explore application areas that are traditionally less accessible to animation, such as glanceable displays.

ACKNOWLEDGMENTS

We thank George Robertson for his comments on a draft of this paper. Thanks also to Steve Drucker.

REFERENCES

1. Assa, J., Caspi, Y., and Cohen-Or, D. Action synopsis: pose selection and illustration. In *Proc. Siggraph'05*, pp. 667-676.
2. Baecker, R., Small, I., and Mander, R. Bringing Icons to Life. In *Proc. CHI '91*, pp 1-6. 1991.
3. Bartram, L. Can motion increase user interface bandwidth? In *Proc. IEEE Conference on Systems, Man, and Cybernetics'97*. pp. 1686-1692.
4. Bartram, L., Ware, C. and Calvert, T. *Moving Icons: Detection and Distraction*, Interact 2001.
5. Baudisch, P., Cutrell, E., and Robertson, G. High-Density Cursor: A Visualization Technique that Helps Users Keep Track of Fast-Moving Mouse Cursors. In *Proc. Interact'03,* pp. 236-243.
6. 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. In *Proc. Interact'03*, pp. 57-64.
7. Bétrancourt, M. and Tversky, B. (in press). Simple animations for organizing diagrams. *International Journal of Human Computer Studies.*
8. Cavanagh, P. and Alvarez, G. Tracking multiple targets with multifocal attention. *TRENDS in Cognitive Science,* Vol. 9. No. 7, pp. 249-354, July 2005
9. Chang, B.-W. and Unger, D. Animation: From Cartoons to the User Interface. In *Proc. UIST'93,* pp. 45-55.
10. Chittaro L., and Ieronutti, L. A visual tool for tracing users' behavior in virtual environments. In *Proc. AVI'04,* pp. 40–47.
11. Feiner, S. APEX: An Experiment in the Automatic Creation of Pictorial Explanations. *IEEE Computer Graphics and Applications,* 5(11), pp. 29-37, 1985.
12. Flash components: download.macromedia.com/pub/documentation/en/flash/fl8/fl8_using_components.pdf
13. Gutwin, C., and Penner, R. Improving Interpretation of Remote Gestures with Telepointer Traces. In *CSCW'05,* pp.49-57.
14. Haller, M., Hanl, C., and Diephuis, J. Non-Photorealistic Rendering Techniques for Motion in Computer Games. In *ACM Computers in Entertainment 2(*4) October 2004.
15. Heer, J., Card, S., and Landay, J. Prefuse: a Toolkit for Interactive Information Visualization. *Proc. CHI'05*, pp. 421-430.
16. Hill, W., Hollan, J., Wroblewski, D., and McCandless, T. Edit wear and read wear. In *Proc. CHI'92*, pp.3-9.
17. Hoobler, N., Humphreys, G., and Agrawala, M. Visualizing Competitive Behaviors in Multi-User Virtual Environments. In *Proc. Viz'04*, pp. 163-170.
18. Kaptelinin, V., Mäntylä, T., Åström, J. Transient Visual Cues for Scrolling: An Empirical Study. In *CHI'02 Extended Abstracts*, pp. 620-621.
19. Klein, C. and Bederson, B. Benefits of Animated Scrolling. In *CHI'05 Extended Abstracts*, pp. 1965-1968.
20. Kurlander, D.J., Skelly, T, and Salesin, D. Comic Chat. In *Proc. SIGGRAPH'06,* pp. 225 - 236
21. Lowe, R. Interactive animated diagrams: what information is extracted? In *Proc. First International Conference on Using Complex Information Systems*, Sept. 4-6 '96, Poitiers, France.
22. Masuch, M., Schlechtweg, S., and Schulz, R. Speedlines Depicting Motion in Motionless Pictures. In *SIGGRAPH'99 Conference Abstracts and Applications,* p. 277.
23. McCloud, S. *Understanding Comics*. Perennial Currents, 1994.
24. McCrickard, S., Catrambone, R., and Stasko, J. Evaluating Animation in the Periphery as a Mechanism for Maintaining Awareness. In *Proc. INTERACT'01,* pp. 148-156.
25. Oksama, L. and Hyöna, J. Is multiple object tracking carried out automatically by an early vision mechanism independent of higher-order cognition? An individual difference approach. *Visual Cognition,* Vol. 11, pp. 631-671, 2004.
26. Reynolds, C. Stylized Depiction in Computer Graphics: Non-Photorealistic, Painterly and 'Toon Rendering. An annotated survey of online resources www.red3d.com/cwr/npr
27. Robertson, G., Cameron, K., Czerwinski, M., Robbins, D. Polyarchy visualization: visualizing multiple intersecting hierarchies. In *CHI'02 Extended Abstracts*, pp. 423–430.
28. Robertson, G., Card, S., and Mackinlay, J. The cognitive coprocessor architecture for interactive user interfaces. In *Proc. UIST'89*, pp. 10-18.
29. Stasko, J. Animation in User Interfaces: Principles and Techniques. In *User Interface Software '93*, pp. 81-101.
30. Terveen, L. and Hill, W. Finding and visualizing inter-site clan graphs. In *Proc. CHI'98,* pp. 448-455.
31. Thomas, B., and Calder, P. Applying Cartoon Animation Techniques to Graphical User Interfaces. In *TOCHI 8(3)*:198–222 (2001).
32. Tversky, B., Bauer Morrison, J. Bétrancourt, M. Animation: can it facilitate? *Int. Journal Human-Computer Studies 57*(4): 247-262 (2002).
33. Ware, C., Neufeld, E. and Bartram, L. Visualizing Causal Relations. In *Proc. INFOVIZ '99*, pp. 39-42.
34. Wickens, C. And Carswell, C. The proximity compatibility principle: its psychological foundation and relevance to display design. *Human Factors 37,* 473–494, 1995.
35. Woodring, J., and Shen, H.-W. Chronovolumes: A Direct Rendering Technique for Visualizing Time-Varying Data. In *Proceedings IEEE TVCG Workshop on Volume Graphics 2003,* pp. 27-34.
36. Yantis, S. Multi-element visual tracking: Attention and perceptual organization. *Cognitive Psych.* 24, pp. 295-340, 1992.