Supporting Information Awareness Using Animated Widgets

D. Scott McCrickard Q. Alex Zhao
Graphics, Visualization, and Usability Center
and College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{mccricks,azhao}@cc.gatech.edu

Abstract

This paper describes a Tcl/Tk widget set extension that supports three animated behaviors: fading, tickering, and rolling. This extension is targeted for use in information awareness applications that monitor and communicate constantly changing information. Described is the programming interface for the widgets as well as the design decisions made in creating them and programs that use them. Introduced in the description are two new techniques called automatic markups and history-based shadowing, highlighting techniques used to identify and communicate the nature of the changes.

1 Introduction

What is animation? Baecker and Small describe it as "sequences of static images changing rapidly enough to create the illusion of a continuously changing picture" [1]. It has been used to generate emotion, to provide entertainment, and, of interest in this paper, to communicate information.

One domain where animation may prove useful is *information awareness*, the need to stay aware of changes to information while accomplishing other tasks. Consider the ever-expanding pool of constantly changing information that is readily available on the Internet and World Wide Web. Stock prices climb and fall, news bulletins arrive, email queues grow, ball teams score points, and the weather changes. While people care about this information, their primary focus is generally on more important tasks: editing documents, programming code, or other job-related activities.

We feel that gradual and repetitive animation can be used to create useful and usable information awareness applications. By animating large amounts of information in a small space, the remaining space can be used for other applications. Changes to the information can be integrated gradually into the display in the next iteration, minimizing the disturbance to the user. Constantly cycling through the entire information space lessens the number of physical interactions required to obtain the information – rather than having to press a series of buttons or keys to get information, the user need only wait for it to cycle through.

To provide easy creation and use of animated effects we have integrated animation support into Tcl/Tk via three widgets. We focused on three effects that seem to meet the criteria discussed earlier: the *fade* widget cycles between items of text, bitmaps, or images, the *ticker* widget scrolls information horizontally across the screen, the *roll* widget scrolls information vertically. Section 2 describes the widgets in more detail.

In creating an animated widget set, numerous design decisions had to be made, not the least of which is the choice of a language. The Web-aware nature of Tcl combined with its powerful string parsing capabilities makes it a good choice for information monitoring applications. The easily extensible widget set and well-defined programming interface of the Tk toolkit provide a good framework for the implementation of animated widgets. Section 3 outlines selected implementation details made in constructing animated widgets, including more reasons behind our choice of Tcl/Tk as the implementation language.

While animation has many potential benefits, it has several drawbacks as well. Animation is often seen as distracting or annoying because of its constant motion and rapid visual changes. Our widget set addresses this problem by giving both the programmer and the end-user significant power over the animation. Another problem with animation specific to awareness is that it can be difficult to see where changes occur and what the previous states of information was. By augmenting the display using information from previous states with techniques we call *automatic highlighting* and *history-based shadowing*, we attempt to lessen the effects of these problems. Section 4 describes in detail our solutions.

Our animated widgets have been available and in use for over a year. The animated widgets have been used by over fifty programmers in a variety of information awareness applications, and some of the programs have been used by more than a hundred users. Section 5 discusses some guidelines for writing information awareness applications based on feedback from both programmers and users.

Overall, we believe that animation is an important technique to have available in a user interface toolkit, and Tcl/Tk is an appropriate language in which to include animation. This paper describes the behavior of the Agentk animated widgets and explores some of the design decisions made in creating them.

2 Animated widgets in the Agentk toolkit

Animation has the potential to enhance the awareness capabilities of interfaces. Our Agentk toolkit contains several interface widgets intended to make it easy for programmers to design awareness applications and easy for people to use them in maintaining awareness of changing information.

Animated widgets are a subclass of *mega-widgets*, a collection of widgets that are operated using a single interface. For example, the LabelText mega-widget in the Tix library provides a single interface to a label widget and an entry widget [8]. Just as mega-widgets augment the power and ease of use of widgets by packaging several of them together as a single new widget, animated widgets augment the behavior of a widget by constantly changing its appearance at regular intervals.

Agentk is a widget toolkit designed to assist in the creation of agent-like programs such as the information awareness applications discussed earlier. Agentk contains three animated widgets: the *fade* widget, the *ticker* widget and the *roll* widget. The fade widget fades between multiple blocks of text, bitmaps, or graphical images. It can be used when the blocks of information have a fixed height and width. The ticker widget horizontally scrolls or "tickers" a stream of text. It can be used for variable length streams of text. The roll widget scrolls text vertically. It is well suited for ordered lists of items.

The programming interface for the fade, ticker, and roll widgets is identical to that for labels, scrollbars, or any other widget: the programmer specifies the widget creation command (ex. `label`, `scrollbar`, `fade`), the position in the display tree (`.fader`, `.dialog.ok`), and possibly some options (`-width 50`, `-fg red`, `-shadowhistory yes`). The options include all those of the standard label widget used to display static text and images, as well as additional options that allow a programmer to show the contents of more than one variable and to control aspects of the animation. In so doing, we provide a programming interface that can be easily understood by a Tcl/Tk programmer.

2.1 Fade widget

Rapid changes in the appearance of a widget often attracts the user's attention to the widget. While this is advantageous in many situations, it could be detrimental in an interface where the users' attention is primarily focused on other tasks. Instead, we need a

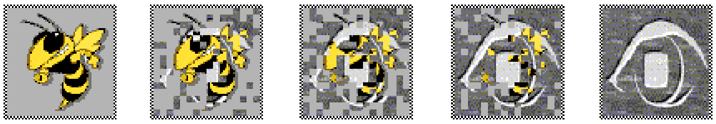

Figure 1: A time-lapse series of the fade widget for two images. Rather than perform compute-intensive calculations to achieve a fading effect, the original image is broken into pieces, and the pieces of the original are gradually replaced with pieces of the final.

widget that can change continuously to match the large and dynamic information space but will change gradually to avoid interrupting the everyday tasks of the user.

The fade widget displays multiple blocks of either text, bitmaps, or graphical images within a given space by gradually fading between them. We expect that the gradual change will be less distracting than a sudden switch yet will allow multiple information blocks to be displayed in a single area. The speed with which the fade occurs can vary depending on the nature of the application: if the widget is designed to attract attention and can be easily stopped by the user, a quick fade might be used, while a secondary display designed to run all the time might call for a slower, less intrusive fade.

The following lines of code give an example of the fade widget displayed in Figure 1.

```
set i1 [image create photo -file buzz.gif]
set i2 [image create photo -file eye.gif]
fade .f -width 100 -height 100 \
  -imagevariable [list i1 i2]
pack .f
.f run
```

The first two lines create two images that are to be faded (actually stippled) in and out. If the values of the variables are changed later, the new images will fade in on the next iteration. The third line creates the widget to display the information contained in the variables `i1` and `i2`. Unlike the textvariable option for other widgets, the fade widget supports a list of variables, fading between each in the list.

2.2 Ticker widget

The ticker widget provides a ticker-tape-style display that scrolls or "tickers" text across the screen. As with the fade widget, a gradual

Figure 2: A news agent that incorporates a fading widget (top right) and tickering widget (bottom right) to alert the user of new stories.

tickering can be less distracting than a sudden switch. The ticker widget has the added advantage that a stream of text could be any length since it never needs to be displayed in its entirety at any given time.

In addition, we take advantage of the natural reaction to grab and pull the ticker widget to make it slow down, stop, back up, or move. When users click and hold down the mouse button within the widget, they can manipulate the information to move in the desired direction and speed. By providing this functionality, the user has more control over the way in which the information is displayed.

The following lines of code gives an example of the ticker widget.

```
ticker .t -width 100 \
  -textvariable [list u1 u2 u3]
pack .t
.t run
```

The first line creates a 100-character-wide widget that is configured to display the information in the variables `u1`, `u2`, and `u3`. The second line packs the ticker widget just as any other widget would be packed, and the third line begins the tickering process.

The ticker widgets have been used in applications to show article headers which, unlike sports scores or stock quotes, can vary widely in length. Figure 2 shows an interface where the ticker widget has been used.

2.3 Roll widget

The roll widget "rolls" or vertically scrolls text across the screen. It seems best suited for a list of items or a columns of information where the order and position of the information is important. The ordering of the list and the positioning of the columns is more apparent in the roll widget than in a ticker or fade widget. As with the ticker widget, the user can grab, hold, and move the roll widget to alter the visible information.

Consider the following quick-and-dirty example for monitoring a printer queue on a Unix system.

```
roll .r -width 60 -height 6 \
   -justify left -font fixed
pack .r
.r run
proc checkq {} {
  .r configure -text [exec lpq]
  after 20000 checkq
}
checkq
```

The first line creates a roll widget that is six lines high and 60 characters wide. The text is left justified and the font is fixed to ensure that the columns will line up. The second and third lines pack the widget and start it running. The `checkq` procedure repeatedly checks the contents of the printer queue using the Unix lpq command and configures the roll widget to reflect the results of this command. Figure 3 shows the result of this script.

System administrators could use this short script to keep an eye on buggy printers or to watch for printer misuse. A user who is printing many documents over a short period of time could extend the script to watch several printers at once to know at any time which is the least busy.

2.4 Options, subcommands, and bindings

This section describes the subcommands, options, and bindings for the Agentk animated widgets.

A subcommand is a command that is available within a widget command. For example, all Tcl/Tk widgets (including the animated widgets) support the `configure` subcommand for querying and altering the widget options. Animated widgets additionally support the following subcommands:

- `run` command starts the animation effect for the widget.
- `pause` command pauses the animation until the run command is reissued.
- `jump` (fade only) jumps to the next item in the fade display list.

A configuration option alters the behavior and appearance of a widget. Common options include `-geometry`, `-font`, and `-width`. The following are some selected options that are available with the animated widgets.

- `-speed` indicates the speed with which the animation runs.
- `-delay` (fade only) is the delay in milliseconds before an item that has faded in begins to fade out.
- `-text[variable]`, `-bitmap[variable]`, and `-image[variable]` control the information that appears in the widget. When the value is changed, the new information animates into view as the old information disappears. For the variable options (such as `-textvariable`), the contents of each variable are animated on the screen. Only one of these options can be used for any given widget (the variable options have highest precedence, and images take precedence over bitmaps, and bitmaps over text).
- `-separator` contains a sequence of characters that separate entries in the ticker and roll widgets. This option is overridden by `-separatorbitmap` or `-separatorimage`, which can contain a bitmap or image separator.
- `-markupstyle` indicates a typeface markup (either bold, italic, or a color) used to highlight changes in information. Thus, if new text appears and the

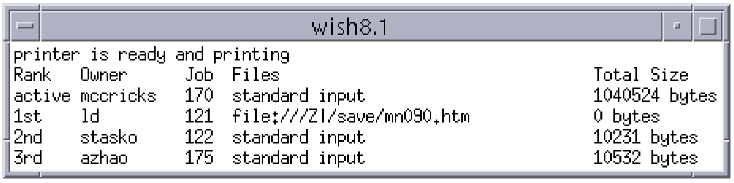

Figure 3: A roll widget that displays the contents of a printer queue. The information is rolled vertically across the screen. The user can grab and move the display if desired and can throw (drag and release) it to adjust the speed.

markupstyle is set to red, the new text will be red. To end the highlighting, `-markupcount` can be set to a number of iterations or `-markuptimeout` can be set to a length of time after which the markups will disappear. Section 4.1 talks more about automatic markups.

- `-shadowhistory` (a boolean) indicates whether history-based shadowing should be enabled. If it is, previous states of the displayed information (text only) are shown using a shadow effect. See Figure 5 for an example. `-shadowcount` and `-shadowtimeout` can be used to remove the shadowing after some number of iterations or some length of time. Section 4.2 talks more about history-based shadowing.

- `-throw` (a boolean) indicates whether speed should be adjusted when user "throws" a ticker or roll widget; that is, drags and releases it.

- `-drive` synchronizes widgets of the same type so that they will run in lock step. For example, the two fade widgets in Figure 4 must always run together as one widget shows images and the other shows labels for the images. As one drives the other, each step in the two fades will be calculated and displayed at the same time.

Bindings define the behavior when certain actions are performed by a user on a widget. Below are the default bindings. A programmer can tailor the bindings based on the needs of their applications.

- `ButtonPress` binding stops the animation effect. Users who want to read the text without being distracted by the animation can press and hold the mouse button.

- `Motion` of the mouse while the mouse button is being held down drags the ticker and roll widgets.

- `ButtonRelease` for the fade widget jumps to the next block of information. For the ticker and roll widget, it calculates a new speed based on the speed at which the widget was dragged. For all widgets it restarts the animation effect.

2.5 Applications using animated widgets

Animated widgets have been used in a variety of programs by over fifty programmers. The figures in this paper show some of the interfaces they have designed. This section focuses on a few interfaces, all of which are available from the Agentk home page given in the Conclusions section.

The NewsAgent interface repeatedly downloads and parses a news Web page looking for new articles (see Figure 2). It alerts users of new articles by changing the color of an image and rapidly fading in and out a "New News" message. The interface contains a ticker widget that continually tickers through the news headers. If a user sees an article of interest, by pressing the image a news viewer can be raised and the user can read the article in its entirety.

The tkscore interface monitors college basketball scores and displays them using a user-selected animated widget or using a standard label widget. We introduced this tool during the NCAA Basketball Tournament, a season ending single-elimination tournament

Figure 4: The fade interface of the CWIC passive Web browser. The display consists of two fade widgets running in sync using the -drive option. The upper one fades between images and the lower one fades between text labels for the image. The first frame shows an initial image and text. The next two frames show how the image and text fade away as the new ones fade in. The final two frames show how the new image and text appear in their entirety as the old disappear.

that generates significant interest and excitement. We asked the users to complete a questionnaire on their informational and animation preferences. The results of this study (available in [9]) have been used to further the development of our animated widgets and have led to the development of the tkwatch interface, a more general tool for monitoring stock quotes, news headlines, weather data, personal information, and sports scores.

The CWIC passive browsing system (see Figure 4) continually browses selected Web sites, identifying key images and presenting them to the user by gradually fading between images. Note that the URL where each image was found is provided as well so that the user can visit the page if an image of interest appears. CWIC is described in detail in [4].

3 Incorporating animation into Tcl/Tk widgets

The style of programming represented by scripting languages is well-suited for information awareness applications. As noted by Tcl creator John Ousterhout, scripting languages are designed to "glue" together existing resources making them easier and more efficient to use [13]. As this closely matches the purpose of awareness applications, (to act as an intermediary to the monitoring of information resources that otherwise may be tedious and time-consuming to do), it seems that scripting languages are an appropriate choice for authoring awareness applications. Tcl/Tk in particular seems to be a good choice for awareness applications. Tcl is designed to be a platform-independent scripting language with an extensible graphical toolkit (Tk) for interface design [12]. Extending this toolkit to include animated widgets provides a solid programming platform for the implementation of awareness applications.

To maintain consistency and decrease the learning time for programmers, animated widgets behave like standard Tk widgets. To simplify this process, the Agentk animated widgets use the Tk requirement library for widget creation, querying, and modification created by Jeffrey Hobbs[1]. The widget package provides a framework for selecting components, creating subcommands and options, and integrating related procedures into a single namespace. Each of the animated widgets has as its sole component the canvas widget. This means that even though the animated widgets appear to behave like a standard label widget, they can make full use of the additional display capabilities of the canvas widget. Initial implementations used a label, but we found that the canvas was necessary to provide all of the desired behaviors.

The Agentk widgets are implemented in Tcl only, meaning that they can be used on any platform with no compilation. The entire package consists of about 4000 lines of code (plus an additional 10000 for the widget package), and individual widgets can be included or excluded as needed to further reduce the overhead. The widgets have been tested on various Unix platforms as well as Windows 95 and 98. All of the widgets can take advantage of the additional image formats and capabilities found in Jan Nijtman's Img extension[2] but it is not necessary to have the extension to use Agentk.

The remainder of this section focuses on two key issues addressed in the creation of

[1] See http://www.hobbs.wservice.com/tcl/script/widget/
[2] See http://purl.oclc.org/net/nijtmans/img.html

the animated widgets in Agentk. The first addresses performance across platforms, particularly important given that slower performance for animations results not simply in longer delays, but in a different appearance.

3.1 Maintaining platform independence

The loop that creates the animation effect (similar for each of the widgets) can be summarized as follows:

```
proc anim {} {
  # calculate next animation step
  ...

  after $delay anim
}
```

The `after` command waits for the period of time specified in the `delay` variable before executing the `anim` command.

Initial Agentk implementations ignored the time required to perform the calculations. On a slower or more heavily-loaded machine, the calculations may take significantly longer, resulting in an animation that is slower. Thus, a program may look very different depending on the platform. To address this problem, we calculated the time required to complete the calculations and subtracted it from the delay. If the resulting delay was less than zero, we adjusted the size of the steps taken by the algorithm. For example, a ticker might move two pixels instead of one, or a fading of text might make larger changes in color over a smaller number of steps.

In so doing, the animated widgets will look similar on any platform, regardless of the processor speed or machine load. The only perceivable difference is that the animation may be smoother on faster, less-heavily-loaded machines. By allowing for this automatic adjustment in performance, a programmer can write a script using animated widgets with confidence that it will appear the same to users on a variety of platforms.

3.2 Calculating an animation step

In fading between blocks of text or bitmaps (see the lower portion of Figure 4), the foreground color of the original information is gradually changed to match the background color, in essence fading the information away. At the same time, the new information (originally "invisible" because it starts as the background color) changes to match the foreground color. When the new information becomes closer (in an RGB-value sense) to the foreground color than the original information, it is raised to the top of the display stack.

When fading between two images (see Figures 1 and 4), it is impractical to change each pixel from the old color to the new – even a small 100x100-pixel image contains 10,000 pixels to repeatedly fade. Instead, the fade widget uses a stippling effect. Each image is divided into small squares, and the squares from the original are replaced with the squares from the new until the effect is complete. The user can specify the size of the squares or can specify the speed with which the animation will occur. If the size is specified, the speed will be dependent on the speed of the machine. A faster machine will be able to calculate and update the display more quickly than a slower one. If the speed is specified, the size of the squares will depend on the speed of the machine. Slower machines divide the images into larger squares, but the time required to fade from one image to another will be the same.

The tickering (and rolling) effects are accomplished by moving all of the items on the canvas horizontally (or vertically). As old textual and graphical items disappear from one side of the canvas, new items are added to the other. The number of pixels by which the display is shifted depends on the response time of the machine. Typically the display is shifted by a single pixel at each step, but slower machines may not be able to keep up with the desired speed at that rate. Our implementation regularly monitors the performance and increases the number of pixels if the system is being taxed. Note that this will result in an animation that is not as smooth, but it is more important that the appearance

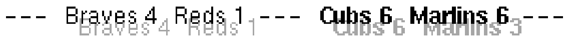

Figure 5: An example of automatic markups and history-based shadowing in a ticker widget showing sports scores. The bold text indicates a score that was recently updated, while the older score appears in plain text. The background shadow shows scores from ten minutes ago.

be similar on all machines.

4 Compensating for animation drawbacks

This paper has discussed ways that programmers can use our toolkit to incorporate animation into their programs in a platform-independent and resource-friendly manner. However, in creating a toolkit it is most important to consider the ability of the user to obtain the desired information. This section discusses ways that the Agentk animated widgets deal with three user-related concerns.

4.1 Highlighting changes with automatic markups

Although animation provides a means to smoothly show the current state of changing information, it can be difficult for a user to identify when and where a change has occurred. To address this problem, the Agentk animated widgets support automatic markups of text. These markups include boldface, italics, and coloring and can occur whenever the information in the widget is updated. For example, Figure 5 shows a widget that displays sports scores that are constantly downloaded from the Web. Recently changed scores are shown in bold text, while older scores appear in plain text.

A protocol is needed for removing the markups from older items. Agentk makes two options available to programmers: an automatic markup can be removed after a given period of time or after a given number of iterations of the display. In addition, the programmer can allow the user to reset the markups with a button press, mouse click, or other action. In our sports scores example, new scores could be highlighted for five minutes, or for ten iterations of the display. The programmer selects the option that seems best suited for the application.

Markups have long been established as a good way to highlight important parts of information. Systems like HtmlDiff [6] have used them to draw attention to changed information. The Agentk toolkit simplifies the integration of markups into situations where it is important to highlight changes. They are instantiated using command options that specify the desired style of markup and when they should be added and removed.

4.2 Showing previous states with history-based shadowing

Sometimes it is not sufficient to simply relate to a user that a change has occurred – it is necessary to communicate information about the nature of the change. To provide this capability, the widgets support *history-based shadowing*, a technique where a previous state of text is shown in the shadow of the current information state (see Figure 5). This technique captures the spirit of integrating supporting material described in [5] framed with a familiar shadow metaphor. In history-based shadowing, the shadow appears slightly offset horizontally and vertically from the original text and appears in a color that is between (in the RGB-value sense) the foreground and background colors. As such, it requires little extra space and is less obvious on first glance than the (more important) current state.

Similar to the automatic markups, the history-based shadowing for a given piece of information appears as soon as changes to the information are noted and can persist for a given length of time, a given number of iterations, or until the next change occurs. This gives the programmer the flexibility to choose a level of persistence that is best suited for the information.

History-based shadowing seems to be a good complement to animation and markups in maintaining awareness of information. The presence and persistence of the shadows is instantiated using Tcl-style command options. Shadowing provides a way to show not only the presence of changes to information, but also the state of the information prior to the changes.

5 Programming guidelines for animated widgets

One of the basic principles of interface design is to help the user maintain a sense of control over the actions on the screen. A user should not have to respond to actions but instead should control the rate at which information is assimilated. Yet animation seems to violate this principle: the flow of information typically is not under the direct control of the user. While our implementation attempts to empower the user by providing bindings for flow control, a programmer must be sensitive to the needs and desires of the user population. In fact, one criticism of animation in general is that it can be disruptive. The "blink" tag in HTML and animated advertisements on Web pages are often appropriately characterized as being annoying and disruptive. However, people have long used computer desktop accessories such as clocks and machine load monitors. What are the distinctions between these situations?

One distinction is that the burden of tolerating the constant changes is outweighed by the advantages these tools provide. Glancing at a desktop clock to obtain the time is far easier than running the date command or even looking at a wristwatch, and looking at a load monitor while running a compute-intensive program is far easier than running commands to determine the system load. A programmer should target application domains where the potential for knowledge is significant and the ability to start and stop the application is easy and apparent.

Another distinction is that the changes to the display are small, subtle, and predictable, allowing the user to adapt to the changing display to the point where it is barely noticeable. We hope that by providing smooth animations (many with user-controllable speeds), the applications will be less distracting. The programmer can assist in this by choosing appropriate sizes and speeds for the application. Users are more likely to use an application if it does not consume much desktop space and if it is not overly distracting.

Our experiences indicate that users are willing to use applications that employ animated widgets, though they generally do not use them continually. We regularly receive comments from users who started up an information awareness application for a few hours, whether it be to keep track of the score of a game in the heat of the playoffs or to watch for new news about the JFK Jr crash, but to our knowledge no one leaves the applications running continually. Programmers should not write applications that employ animation with the expectation that they will be used continually, but rather for short, well-defined periods of time, perhaps to monitor the traffic 5 and 6 pm every weekday, or to keep an eye on the scores of selected baseball games during the pennant drive and playoffs. If programmers do anticipate that it will be necessary to run the application at all times, alternate (non-animated) information delivery mechanims should be made available.

6 Related work

Animation has been used in various information visualization situations. Baecker and Small list several uses for animation, including among others identification, transition, demonstration, history, guidance, and feedback [1]. Some examples include the percent-done indicators for providing feedback [10] and animated icons for demonstrating use [2]. One use that has not been explored to the fullest is the application of animation to better utilize screen space. Cone trees [15] are an example of a visualization in which animation is used to show more information (in this case about hierarchies) than would otherwise be possible in a given space. We feel that this is a distinct advantage in information awareness applications.

Animation has been used to show dynamic information as well. Algorithm animation systems demonstrate how the dynamic processes of algorithms work [16]. The Tickertape interface scrolls text messages from email and other resources across the screen [14]. Bartram suggested the use of animation as an additional display dimension in the design of interfaces as well [3]. Yet, little work has been done on continuous, long-term animation for secondary awareness tasks. Animation has been dismissed as too intrusive, but here we argue that it can be used in some situations to maintain awareness without being too distracting.

Several other widget sets have been developed that allow the programmer to include animation in the interface. The Artkit toolkit allows programmers to create transition objects that describe how an object will move [7]. A reference to the transition object is then added to a graphical object to create the animation. Another toolkit, Amulet, was extended to include support for animation [11]. Animation can be attached to any value of any object, including position, color, or visibility. These and similar toolkits provide a great deal of power, yet they often can require significant effort by the programmer to achieve a desired result. The widgets in Agentk do not require the user to specify the details of the animation. In only a few lines of code, a programmer can specify variables, define animation behavior, and start a cyclic, repetitive animation that automatically updates when variables are changed by the program. This same behavior would be much more difficult with most other toolkits. At the same time, the Agentk animated widgets are structured such that new animated widgets can be added with significant code reuse.

7 Conclusions and future work

This paper has discussed the integration of animation into the Tcl/Tk toolkit. Incorporating animation and related techniques into a user interface toolkit makes it possible for programmers to include it in their applications using a familiar programming interface and makes it easier for users to keep a sense of control while maintaining a desired level of information awareness.

Our future work will concentrate on three areas. First, we plan to extend widget set to include other behaviors such as shrinking and growing, swiping, and others. Second, we are considering adding other optional techniques such as motion trails and slow-in/slow-out that may lessen the distraction caused by the animations. Third, we hope to make the execution more efficient, by making use of simplified calculation methods, improvements in machine speed calculations, and perhaps threading of the widget commands.

Programmer and user reaction to animated widgets has been encouraging, and we expect that the use will continue as the widget capabilities increase. The Agentk toolkit, including the animated widgets and most of the programs described in this document, is available at `http://www.cc.gatech.edu/~mccricks/agentk`.

Acknowledgments

The authors would like to thank the USENIX Association for providing several grants that made this work possible. We would also like to thank the GVU Center and the College of Computing at Georgia Tech for their support via grants, lab space, and machines. Finally, we would like to thank John Stasko and the Information Interfaces Group for their helpful comments on the Agentk toolkit and on this document.

References

[1] Ronald M. Baecker and Ian Small. Animation at the interface. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design*, pages 251–267. Addison-Wesley, 1990.

[2] Ronald M. Baecker, Ian Small, and Richard Mander. Bringing icons to life. In *Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems*, pages 1–6, New Orleans, LA, May 1991.

[3] Lyn Bartram. Enhancing visualizations with motion. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '98)*, pages 13–16, Raleigh, NC, 1998.

[4] Quasedra Y. Brown and D. Scott McCrickard. Cwic: Using images to passively browse the web. Technical Report 99-48, Georgia Tech GVU Center, Atlanta, GA, 1999.

[5] Bay-Wei Chang, Jock D. Mackinlay, Polle T. Zellweger, and Takeo Igarashi. A negotiation architecture for fluid documents. In *Proceedings of the ACM Symposium on User Interface Software and*

Technology (UIST '98), San Francisco, CA, November 1998.

[6] Fred Douglis, Thomas Ball, Yih-Farn Chen, and Eleftherios Koutsofios. The AT&T internet difference engine: Tracking and viewing changes on the web. *World Wide Web*, 1(1):27–44, January 1998.

[7] Scott E. Hudson and John T. Stasko. Animation support in a user interface toolkit: Flexible, robust, and reusable abstractions. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '93)*, pages 57–67, Atlanta, GA, November 1993.

[8] Ioi K. Lam. Designing mega widgets in the Tix library. In *Proceedings of the 3rd USENIX Tcl/Tk Workshop*, Toronto, CA, July 1995.

[9] D. Scott McCrickard, John T. Stasko, and Q. Alex Zhao. Exploring animation as a presentation technique for dynamic information sources. Technical Report 99-47, Georgia Tech GVU Center, Atlanta, GA, 1999.

[10] Brad A. Myers. The importance of percent-done progress indicators for computer-human interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (SIGCHI '85)*, pages 11–17, April 1985.

[11] Brad A. Myers, Robert C. Miller, Rich McDaniel, and Alan Ferrency. Easily adding animations to interfaces using constraints. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '96)*, Seattle, WA, November 1996.

[12] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA, 1994.

[13] John K. Ousterhout. Scripting: Higher level programming for the 21st century. *IEEE Computer*, March 1998.

[14] Sara Parsowith, Geraldine Fitzpatrick, Bill Segall, and Simon Kaplan. Tickertape: Notification and communication in a single line. In *Asia Pacific Computer Human Interaction 1998 (APCHI '98)*, Kangawa, Japan, July 1998.

[15] George G. Robertson, Stuart K. Card, and Jock D. Mackinlay. Information visualization using 3d interactive animation. *Communications of the ACM*, 36(4):57–71, April 1993.

[16] John T. Stasko. The path-transition paradigm: A practical methodology for adding animation to program interfaces. *Journal of Visual Languages and Computing*, 1:213–236, 1990.