Snap-and-go: Helping Users Align Objects Without the Modality of Traditional Snapping

Patrick Baudisch, Edward Cutrell, Ken Hinckley, and Adam Eversole
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
{baudisch, cutrell, kenh, adame}@microsoft.com

ABSTRACT
Snapping is a widely used technique that helps users position graphical objects precisely, e.g., to align them with a grid or other graphical objects. Unfortunately, whenever users want to position a dragged object *close* to such an aligned location, they first need to deactivate snapping. We propose *snap-and-go*, a snapping technique that overcomes this limitation. By merely stopping dragged objects at aligned positions, rather than "warping" them there, snap-and-go helps users align objects, yet still allows placing dragged objects anywhere else. While this approach of inserting additional motor space renders snap-and-go slightly slower than traditional snapping, snap-and-go simplifies the user interface by eliminating the need for a deactivation option and thereby allows introducing snapping to application scenarios where traditional snapping is inapplicable. In our user studies, participants were able to align objects up to 138% (1D) and 231% (2D) faster with snap-and-go than without and snap-and-go proved robust against the presence of distracting snap targets.

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

Keywords: Alignment, snapping, snap-dragging, mouse input, pseudo haptics.

INTRODUCTION
Sometimes users need to perform *precise* graphical manipulations. As shown in Figure 2, users increase visual clarity by aligning graphical objects with each other or by scaling table columns to the same width. They align audio and video segments to assure synchronicity and they make precise selections in bitmap images to make sure subsequent filters get applied to the right areas.

Obtaining precise results by dragging or scaling an object with the mouse requires considerable motor skill. Therefore, many computer applications help users align objects. *Snapping* (e.g., *snap-dragging* [5]) provides aligned positions with an attraction behavior sometimes described as "magnetism" [3] or "gravity" [5]. Figure 1a illustrates this with an example of a slider with a single snap location.

CHI 2005, April 2–7, 2005, Portland, Oregon, USA.

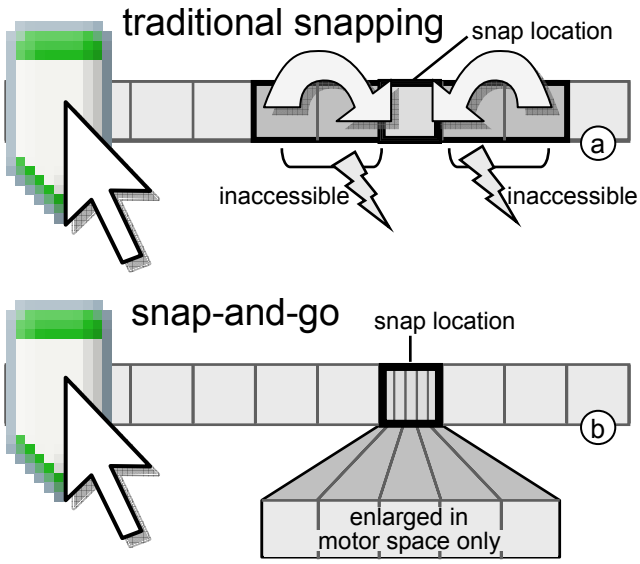

Figure 1: (a) The problem: Traditional snapping warps the knob of this slider to the target whenever close, making it impossible to place the knob in the areas marked inaccessible. (b) The proposed solution: Snap-and-go inserts additional motor space at the snap location, thereby keeping all slider positions accessible.

Whenever the user drags the knob into the area surrounding the snap location the knob is automatically "warped" to the snap location. Given that this attraction area is larger than the snap location itself, here five pixels instead of one, the alignment task is simplified significantly.

The downside of snapping, however, is that this magnetic behavior can get in the user's way. While users are often likely to use the recommended snap locations, there are cases where users want to place a dragged object elsewhere. Continuing the examples from Figure 2: (a) To align the *baselines* of these text fragments, a user may need to move the right one down *a little*, but the grid keeps holding it back. (b) To fit a slightly longer word into this table cell, the user may need to widen that column just a bit, but it scales in steps causing the next column to overflow. (c) The user tries to make space to allow the following voice over clip start *a little before* the matching video segment, but scaling either snaps back or scales too much. (d) To remove the frame around this picture, the user tries to create a selection around it, but the lasso either snaps to the entire image or leaves at least two pixels out. In these and other scenarios, traditional snapping prevents users from accomplishing their task, every time the user tries.

In the following sections, we take a closer look at why this is problematic; present the *snap-and-go* alignment technique and explain how it avoids this problem; go over the related work; and report three user studies in which snap-and-go was found to improve participants' speed when aligning objects on the screen. We conclude with a summary of our findings and an outlook on future work.

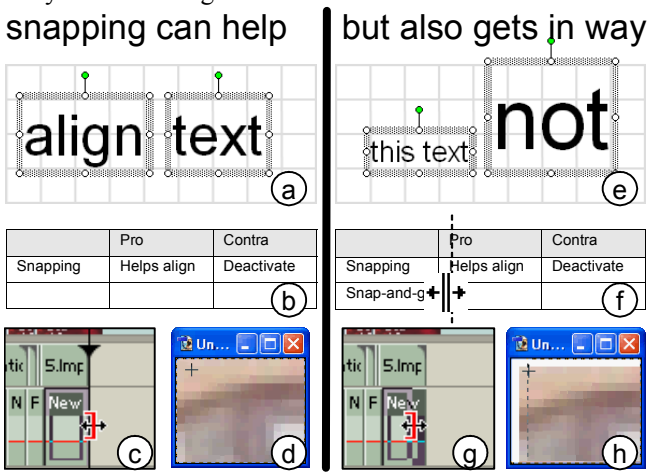

Figure 2: Traditional snapping helps (a) align objects in a graphics editor, (b) created equally sized columns, (c) align audio clips with video clips, and (d) assure that the entire bitmaps was selected, (e-h) but gets in the way when users editing goals are not foreseen by the application designer.

Traditional snapping requires deactivation

In order to enable the *non*-alignment tasks above, traditional snapping requires application designers to provide snapping objects with an additional user interface that allows users to *disable* snapping. This, however, turns out to be more complex than expected.

A common approach is to allow users to de/activate snapping by holding down a modifier key. However, the modifier key approach is inapplicable if (1) the application is already using all modifier keys for other purposes, such as for switching tool options (e.g., Adobe Premiere), (2) there are more snapping constraints than modifier keys (e.g., Microsoft Visio), or (3) the target audience are non-experts and cannot be expected to discover modifier keys.

In these cases, application designers are forced to revert to offering on-screen controls, such as a checkbox in a context menu (Figure 3a). To improve discoverability, some application designers place checkboxes right into the visible user interface, even though this adds to the complexity of the interface (Figure 3b). In case there are too many snapping options, deactivation ends up in an options dialog (Figure 3c), again with low discoverability.

As the participants of our user study confirmed, snapping is a useful and highly appreciated function. It saves users time with every use and enables users with limited motor skills to perform tasks they otherwise could not perform at all. The price, however, is additional user interface complexity—potentially an additional checkbox per snapping function and time spent using it. *Snap-and-go* is designed to overcome this limitation.

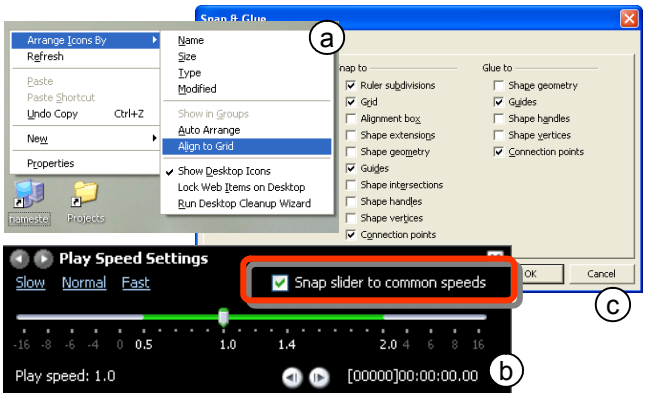

Figure 3: Checkboxes for deactivating snapping in (a) Windows XP, (b) Media Player (c) and Visio.

SNAP-AND-GO

Snap-and-go is a snapping method that does not require deactivation. Figure 1b illustrates this with the example of a slider. Instead of reassigning motor space to snap locations, snap-and-go inserts *additional* motor space at the snap locations. This is done by reducing the input gain of the mouse while over the target. Rather than seeing the knob jump to the target, users feel that the mouse temporarily stops at the target—thus the name of the technique, derived from the expression *stop-and-go*. Visual feedback (e.g., Figure 14b) informs users about successful alignment.

Unlike traditional snapping, snap-and-go does not require deactivation to allow users to avoid alignment. It achieves this by managing motor space differently. Traditional snapping requires a deactivation option because its approach of *redistributing* motor space to the snap location leaves other locations with no representation in motor space, which renders these locations inaccessible. Snap-and-go's approach of *inserting* additional motor space at the snap location leaves the motor space of all other locations intact.

Replacing traditional snapping with snap-and-go allows users to enjoy snapping functionality without the need to learn about modifier keys or to sacrifice screen space for checkbox interfaces (Figure 3). In addition, snap-and-go can be applied to applications that do not offer snapping today, such as to help users center audio balance or to drag the time slider in a DVD player to the beginning of a chapter. These scenarios have no space for a deactivation interface, which is why traditional snapping has never been applied to them. In its current version, snap-and-go is limited to platforms using an *indirect* input device—*direct* input devices, such as pens or touch input do not allow creating extra motor space.

SNAP-AND-GO IN 2D

The slider example given in the previous section is limited to one-dimensional drag interactions. Snap-and-go in 2D is similar to the 1D case in that they both insert additional motor space to hold dragged objects at aligned locations. In addition, however, alignment in 2D requires *guiding* dragged objects to aligned positions. While dragging an object in 1D will inevitably cross all locations between

start and end position, dragging an object in 2D traverses only *one* path and therefore cannot guarantee that a dragged object will ever get to the aligned position in the first place. Since warping was the reason why traditional snapping required deactivation, we need an alternative mechanism for bringing the dragged object to the target.

Snap-and-go in 2D is based on two basic widgets that application designers can overlay over screen objects to help users align objects with them. Similar to the 1D case, these widgets manipulate motor space, but they contain additions to guide dragged objects to snap locations. To provide a basis for explaining the actual snap-and-go widgets (Figure 5), we start by looking at the underlying concepts and evolution history. These are illustrated by Figure 4.

(a) The problem: The user's task is to align the partially visible gray square at the bottom right with the two fully visible gray squares. This requires placing the square at the location marked with a dashed outline. The user's drag direction (shown as a black arrow) would miss the aligned position.

To make the following steps easier to illustrate, we switch to the visuals of a target acquisition task. We paint a knob labeled onto the dragged object and a matching socket labeled onto the position where the knob has to go in order to align the objects. Both are only for the purpose of illustration and are not visible to the user.

(b) We add an invisible "funnel" over the socket. The "funnel" consists of two invisible guides. As the user drags the square, the knob now hits the funnel and slides along that funnel until the knob gets trapped at the socket in the funnel's center and the user is provided with visual feedback confirming alignment (e.g., Figure 18).

The funnel widget simplifies 2D alignment for two reasons. First, instead of having to simultaneously steer the mouse to precise x and y coordinates, the funnel widget requires users to only hit the entrance of the funnel and to follow through. The funnel thereby turns the 2D acquisition task the user would normally have to perform into a 1D acquisition task, also known as "crossing task" [1]. Second, the new target of the user's task, i.e., the entrance of the funnel is significantly bigger than the single-pixel aligned position that the user would otherwise have to acquire. Application designers can configure this width by choosing a funnel of appropriate size.

(c) Aligning the funnel and extending it into a cross. Dragged objects reach the funnel center fastest when making contact with the funnel at a more obtuse angle. While an obtuse angle can be guaranteed by rotating the funnels towards the mouse pointer, we choose a stationary funnel aligned with the coordinate system of the screen. The benefit of this is that it helps users align the dragged object in x or y or both, which is often useful. Duplicating the funnel for all four directions forms a *cross widget* (made of a horizontal and a vertical *guide*), which allows users to align objects dragged when coming from different directions.

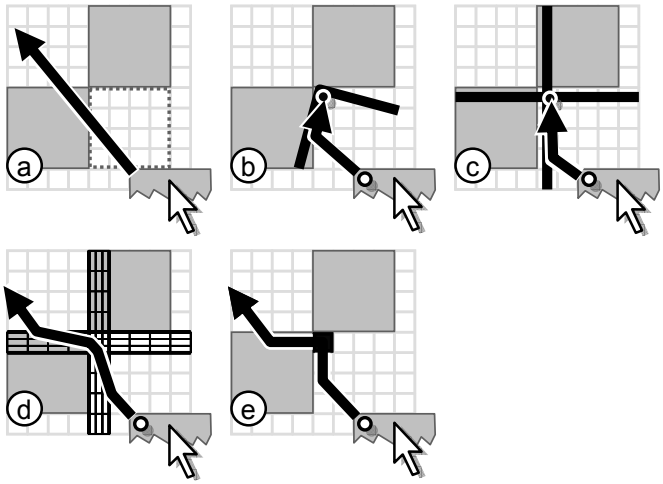

Figure 4: Evolution of snap-and-go in 2D

(d) Minimizing side-effects by using 'frixels': When trying to drag an object to a *different* location on the screen, users may accidentally hit a cross widget on the way. To minimize disturbance, we replace the solid guides of the cross widget with permeable ones. These are created using pixels that slow drag motion down—we call this *friction*. To obtain the desired guidance effect we need pixels with *directional friction* (or "*frixels*"). The vertical guide in this example consists of frixels with a horizontal friction of 3 and a vertical friction of 1, shown as pixels subdivided into three thinner subpixels. Crossing such a frixel with the mouse requires the user to move the mouse horizontally three times as far than a normal pixel would. This causes the path of the dragged object to be *bent*, guiding it through the snap location at the funnel center. At the funnel center, guides overlap and their friction values add up, here resulting in a center frixel with 3x3 friction. This frixel holds the dragged object for a moment, which helps users release it.

(e) Frixel-based widgets create visual effects similar to solid widgets. To obtain a precise trajectory, snap-and-go tracks the position of the dragged object in terms of subpixels. This extra information, however, is not presented to the user; a user dragging an object through a frixel widget sees the dragged object progress in terms of complete pixels. Figure 4e shows how the trajectory from Figure 4d appears to the user as a series of discreet events: the dragged object latches onto the vertical guide, it slides up two pixels where it reaches the aligned position, gets held there for a moment, and then slides along the horizontal guide for two pixels until it breaks free.

We are now ready to explain the two widgets that form the basis of snap-and-go in 2D. The *plus widget* shown in Figure 5a is the basic widget for simultaneous alignment in x and y. It is based on the cross widget from Figure 4d, but it is of finite size and its friction is faded out towards the periphery to minimize the impact on trajectories not aiming for this target. The *bar widget* shown in Figure 5b is designed for alignment in only one axis. It is made of traditional, non-directional friction, which avoids the here unnecessary sideway-motion that directional frixels introduce.

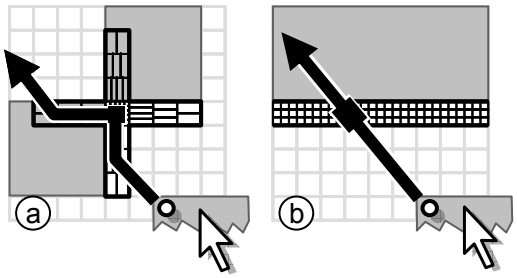

Figure 5: The basic widgets of snap-and-go in 2D are (a) plus widget and (b) bar widget.

More complex widgets can be created by combining basic widgets. The example shown in Figure 6a helps users snap a connector line to a rectangle on screen. While the corresponding object based on traditional snapping (Figure 6b) allows users to connect *only* to edge centers (which can lead to undesired "elbows"), the snap-and-go widget allows users to *also* connect anywhere along the edge.

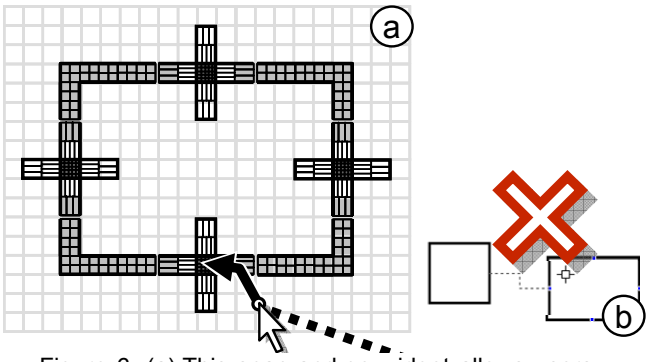

Figure 6: (a) This snap-and-go widget allows users to connect a connector line to edge centers, but also anywhere else on its edges. (b) The same widget based on traditional supports only edge centers.

The example widgets shown in this section feature friction values between 2 and 5. We chose these values to assure the readability of the diagrams. While friction values that low are possible (and we found significant effects for friction values as low as two), we found most users to prefer higher friction values (20-30, see the user study sections).

RELATED WORK

Snap-and-go builds on three areas of related work, i.e., alignment techniques, constraints and snapping, and manipulation of motor space/mouse gain.

Alignment techniques in related work fall into two main categories. The first category contains techniques that are applied post-hoc, similar to traditional menu or toolbar-based alignment functions: users pick two or more objects and then choose a function, such as "stack vertically" from a toolbar. The Alignment Stick [23] allows aligning objects by pushing a ruler against both objects—the moment the second object starts moving both objects are aligned. This approach can reduce the need for selecting alignment functions repeatedly and thus save user effort.

The second category consists of techniques that are applied while dragging the object to be aligned. The original snap-dragging technique by Bier [5] allows users to create and place alignment objects; subsequently placed graphical objects then automatically snap to these alignment objects. By avoiding the need for an extra interaction step, snapping eliminates the overhead faced by explicit alignment functions. Various researchers have added to the concept of snap-dragging by extending it to 3D [6, 2], adding anti-gravity feedback to inform users when attempting to create an illegal connection [13], or changing snapping grids while dragging objects (*HyperSnapping* [20]). Snapping has been applied to a wide range of applications, including snapping and zipping windows together [4]. A particularly simple and thus widely used alignment object is the grid. The *CAGE* [2] extends grids such that they allow aligning graphical objects with *each other*.

Another way of aligning objects is to describe the desired goal state using constraints [8]. Constraints are supported by a variety of toolkits, such as *Juno* [22] and their use reaches back as far as to Sutherland's *Sketchpad* [24]. While initially created with text interfaces, some systems allow users to established constraints using snap-dragging (*augmented snapping* [11]). Similarly, [3] allows users to manipulate aligned groups without giving up alignment. Other researchers suggested creating alignment behavior *by demonstration* [18] or by generating several aligned versions and letting users pick (*suggestive interfaces* [14]).

Snapping and constraints restrict the space where objects can be placed. Since this leads to the aforementioned problem of inaccessible space, snap-and-go inserts additional motor space instead. Manipulation of motor space has been studied in the field of *pseudo haptics* [17] and can be applied to any indirect pointing device. Lécuyer et al. showed that changing the coupling between mouse motion and mouse pointer motion can be used to simulate the haptic sensation of texture [16].

Changes in the mouse-to-pointer gain have also been used to help users overcome long distances and to acquire small targets, e.g., *object pointing* [12], and *lay lines* [15]. Expanding targets (in screen space and motor space) [21] was found to help users acquire small targets. In combination with an *area cursor*, making targets "sticky" was found to help users with motor disabilities acquire small targets with the mouse (*sticky icons* [27], *semantic pointing* [7], also suggested by [25]). Unlike these methods, snap-and-go offers a method for guiding the user to very small targets, as we will discuss in more detail at the end of the following section.

SNAP-AND-GO FOR TARGET ACQUISITION

Target acquisition techniques are relevant to the topic of alignment, because an alignment operation can be reduced to a target acquisition task, (e.g., the "Snap-and-go in 2D" section above). Based on this similarity, we created an adapted version of snap-and-go that serves as a target acquisition aid. As shown in Figure 7a, the plus widget remains the same, only the visuals change: Rather than guiding a *dragged object* to an *aligned position*, the cross wid-

get now guides the *mouse pointer* to the *target*. Note that plus widgets always guide the pointer to the target center, thus work across target sizes (Figure 7b).

The snap-and-go target acquisition technique offers benefits similar to the snap-and-go alignment technique: it allows users to distinguish between multiple targets in close proximity, where other techniques, such as snap-to-target or area cursor [27] fail to distinguish between them.

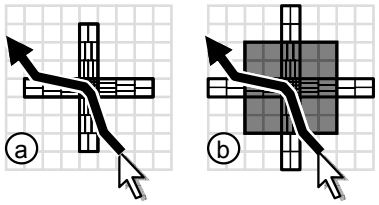

Figure 7: Snap-and-go as a target acquisition aid.

Note that the inverse is not true: a technique originally designed to be a target acquisition technique *cannot* necessarily serve as an alignment technique. Enhancing an alignment position with a sticky icon as shown in Figure 8a turns the acquisition task into a crossing task, which can simplify target acquisition [1]. Crossing that pixel, however, is still hard. Note that sticky icons cannot be used to make a cross widget (Figure 8b); their *non*-directional friction will *not* guide the dragged object to the target. Figure 8c and d illustrate the difference: A sticky icon measuring only one pixel is easily missed. Snap-and-go guides the pointer or dragged object into the target, thereby creating a fisheye effect in motor space.

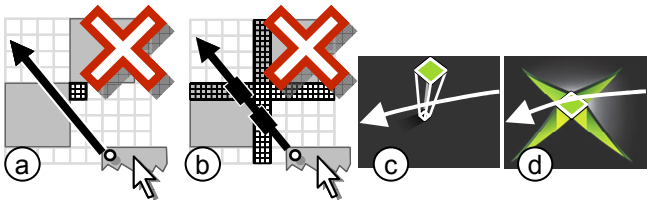

Figure 8: (a, b) Attempt to use sticky icons to align objects in 2D. (c) Sticky icons vs. (d) snap-and-go

IMPLEMENTATION

We created two implementations of snap-and-go, i.e., a complete implementation in C# and a reduced prototype in Macromedia Flash that we use for running user studies. The Flash version supports 1D snap-and-go and the simplified version of 2D snap-and-go shown in Figure 4d. Figure 9 illustrates how this prototype implements snap-and-go by subtracting friction at snap locations from traversed distances. A simple 2D cross widget can be obtained by running this code on *x* and *y* coordinates. (Note that this code is abbreviated for space reasons; it misses code for updating the mouse pointer to keep knob and pointer together, etc.).

The C# version supports the more advanced versions of 2D snap-and-go described in this paper. The code is based on rectangular friction objects, each of which defines a friction gradient of configurable direction and strength. By combining multiple friction objects application designers can create arbitrary friction widgets, including the plus and the bar widgets and the example shown in Figure 6. By integrating friction along the interpolated pointer path, the program assures all traversed friction widgets will take effect, even in cases where the stepwise nature of mouse motion causes the pointer to jump over a widget without actually touching it. Pointer position is tracked in subpixels. This assures that users can traverse friction widgets even very slowly and at flat angles. Rounding errors would otherwise cause progress across the widget to continuously be rounded to zero, causing frixel widgets to appear solid.

```
snapTo(x, w, snapX) {
    if (snapAndGoActive) {    // snap-and-go
        if (x >= snapX + w)
            return x - w + 1;
        else if (x > snapX)
            return snapX;
        else return x;
    } else {                  // traditional snapping
        if (x > snapX - w/2 && x < snapX + w/2)
                return snapX;
        else return x;
    }
}
```

Figure 9: Code fragment for 1D snap-and-go with a single snap location of width w located at snapX (top) in comparison to traditional snapping (bottom). The function returns the location of the dragged knob in dependence of the pointer position.

USER STUDIES

To objectively evaluate performance using snap-and-go, we performed a series of three user studies. The participants' task in all three studies was to align a dragged object with a target location with pixel-accuracy. The studies differed in whether there was a single attractor at the target or multiple attractors, and whether alignment took place in one or two dimensions (Table 1).

	Snap-and-go compared to traditional snapping…	Snap-and-go with distractors…
…in 1D	Study 1	Study 2
…in 2D	Study 3	

Table 1: Scope of the 3 studies reported below

USER STUDY 1: SNAP-AND-GO VS. SNAPPING IN 1D

The purpose of this first study was to verify that snap-and-go indeed helps users align objects, to explore the impact of attractor strength on task time, and to compare snap-and-go with traditional snapping.

Task

The participants' task was to drag the knob of a slider to a highlighted target location as quickly as possible. Figure 10 shows the apparatus, which consisted of a horizontal slider with a single highlighted target location, which in some conditions was complemented with an *attractor* (see below). For each trial the slider was reinitialized to the shown state; target distance and attractor, however, were varied.

Task time was counted from the moment the knob was picked up until the moment the knob was successfully aligned and the participant had released the mouse button. Each trial required successful alignment, so in cases where participants released the knob anywhere but over the target, they needed to pick it up again to complete the trial.

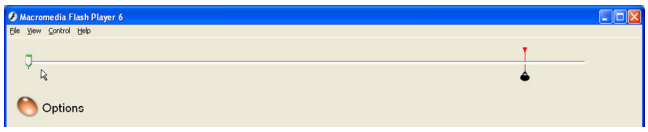

Figure 10: The apparatus. The user's task was to align the slider knob located at the left with the target located at the right.

Alignment required pixel precision. To make that possible the knob was provided with the visuals shown in Figure 11a. To prevent the mouse pointer from occluding the target, participants were encouraged to drag the mouse slightly downwards while dragging the knob (Figure 11c).

Interfaces

There were three main interface conditions, namely *traditional snapping* and *snap-and-go*, implementing the two snapping functionalities illustrated by Figure 1, as well as no snapping.

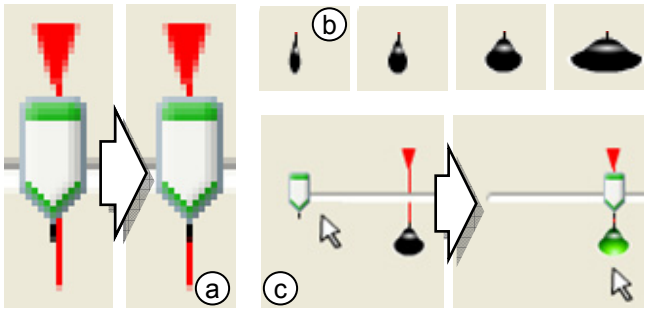

Figure 11: Close-up of the knob reaching the target: A black dash at the bottom of the knob helped visually verify alignment. (b) Attractors used "light bulb" visuals and came in four sizes. (c) Dragging the knob into an attractor caused it to light up.

In the no snapping condition, the target consisted only of the vertical red line shown in Figure 11a. In the snapping conditions, the target was complemented with an attractor, turning the target into a snap location. Attractors behaved differently depending on the snapping condition, but offered the same visuals, a "light bulb" located below the slider (Figure 11b and c). In their inactive state light bulbs were black, but turned to bright green when the knob was captured. To inform participants during the study about the current attractor strengths, the width of the bulb on screen reflected the width of the attractor in motor space (Figure 11a). Interface conditions thus differed in interactive behavior and visuals.

Experimental design

The study design was within subjects 2 x 4 x 4 (Snapping Technique x Attractor Width x Target Distance) with 8 repetitions for each cell. Distances were 100, 200, 400, and 800 pixels, and Widths 5, 10, 18, and 34 pixels. In addition, participants performed 2 blocks of trials with snapping off at each distance. For each trial, we recorded task completion time and error, i.e., number of times the participant dropped the knob before aligning it properly. Interface order, distances, and sizes were counterbalanced.

Participants received training upfront and at the beginning of each block. The study took about 35 min per participant.

Apparatus

The experiment was run on a PC running WindowsXP with an 18" LCD monitor, at a resolution of 1280x1024 pixels and 60Hz refresh rate, and driven by an nVidia graphics card. The interface used in this study was implemented in Macromedia Flash; its functioning was briefly described in the Implementation section of this paper. The optical Microsoft IntelliMouse was set to a medium mouse speed and participants were allowed to adjust it prior to the beginning of the study.

Participants

Nine volunteers, (7 male) between the ages of 25 and 50 were recruited from our institution. Each received a lunch coupon for our cafeteria as a gratuity for their time. All had experience with graphical user interfaces and mice; three were trackball users. All were right-handed.

Hypotheses

We had three hypotheses: (1) Participants would perform faster with snap-and-go than with no snapping. (2) Stronger attractors and shorter distances would reduce task time. (3) Due to the additional distance in motor space, participants should be slightly slower when using snap-and-go then when using traditional snapping. However, we expected the difference to be small.

Results

To correct for the skewing common to human response time data we based our analyses on the median response time across repetitions for each participant for each cell.

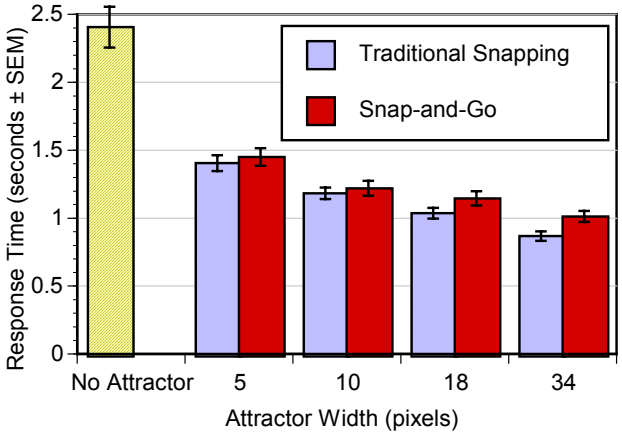

Figure 12: Task time by snapping technique and attractor width across all distances (+/- standard error of the mean).

Snapping vs. no snapping: We compared the most conservative case for the two snapping conditions (attractor size = 5) against no snapping for each distance. We performed a 3 x 4 (Snapping Technique x Target Distance) within subjects analysis of variance. There were significant main effects for both Snapping Technique, $F(2,16)=66.3$, $p<<0.01$ and for Target Distance, $F(3,24)=20.19$, $p<<0.01$. Planned comparisons of no snapping vs. traditional snapping and vs. snap-and-go were also significant, $F(1,8)=76.7$, $p<<0.001$ and $F(1,8)=61.5$, $p<<0.01$ respectively.

Snap-and-go vs. traditional snapping: We performed a 2 x 4 x 4 (Snapping Technique x Attractor Width x Target Distance) within subjects analysis of variance. We found significant main effects for each factor. As expected, traditional snapping was faster than snap-and-go $F(1,8)=24.0$, $p<0.01$. Also as expected, differences were fairly small, ranging from 3% for attractor widths 5 and 10 to 14% for attractor width 34 (Figure 12)

Impact of attractor width and distance on task time: Not surprisingly, there were significant effects for Attractor Width, $F(3,24)=97.6$, $p<<0.01$ and for Target Distance, $F(3,24)=224.4$, $p<<0.01$; performance improved as attractor width increased and as target distance decreased. There were no significant interactions. Given the similarity to Fitts' Law experiments, we compared user performance against the Fitts Index of Difficulty (ID), a metric that combines target width and movement distance into one measure of acquisition difficulty [19, 10]. Figure 13 plots mean movement time for each ID for the two snapping techniques. The regression of movement time against ID for each snapping technique was:

Snapping: $MT=0.265 + 0.19*ID$, $r^2=0.75$

Snap-and-go: $MT=0.487 + 0.159*ID$, $r^2=0.59$

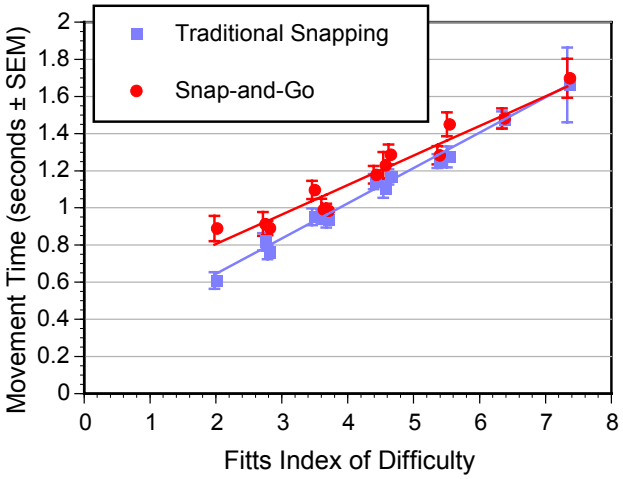

Figure 13: Fitts analysis of task times.

Note that the main divergence is at very low indices of difficulty. Once the task gets harder (e.g., longer movements, smaller attractor sizes) performance in the two techniques begins to converge.

Error rates were generally low, indicating that the target and knob visuals did allow participants to visually validate alignment sufficiently well. Differences in the error rates for the three different snapping conditions were non significant (No Snapping: 6.1%, Traditional Snapping: 3.7%, Snap-and-Go: 2.6%).

Across snapping methods, eight of nine participants indicated a preference for the stronger 34 and 18 width attractors; one participant preferred the weakest attractor strength included in the study (5).

USER STUDY 2: IMPACT OF DISTRACTORS

Experiment 2 was designed to investigate two questions. First, in the case of multiple potential targets (e.g., chapters along a DVD player time slider) should an application designer provide each chapter with an attractor or would the additional attractors (*distractors*) get in the user's way? Second, even with multiple attractors being present, users might at least occasionally want to target a non-enhanced location. How will distractors affect that?

Task and interface were the same as in User Study 1, except for the following two differences. First, in half of the trials there was an attractor over the target, while in the other half there was not. Second, there were up to four "distractors" located in front of and behind of the target as shown in Figure 14 (distances 64, 32, and 16 pixels in front, and 16 pixels behind target, see Figure 14).

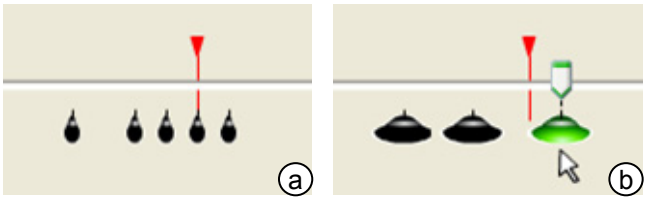

Figure 14: Attractor/distractors: (a) Attractor at target and all four distractors (width 10). (b) No target attractor, but three of the four distractors.

In this study, we only included snap-and-go but *not* traditional snapping. The reason is that for some distractor combinations traditional snapping would have required deactivation (e.g., Figure 14b) and deactivation interfaces were outside the scope of this study.

To keep the overall study time manageable, we limited the design to a single distance and two attractor widths. The resulting design was a (2 x 2 x 2 x 2 x 2 x 2) (Target Attractor on x Distractor 1 x Distractor 2 x Distractor 3 x Distractor 4 x Attractor Width) with 4 repetitions for each cell.

Nine participants hired from the community (6 male) between the ages of 18 and 60 participated in the study. All were right-handed.

Results

The full factorial analysis was very difficult to interpret because of interaction effects. Initial results showed no significant effect for distractor position, so we simplified the analysis by combining the four binary Distractor vari-

ables into a single variable, Number of Distractors (0, 1, 2, 3, or 4). We then did two analyses: one for when the Target Attractor was on, and one for when it was off.

Target with attractor: We performed a 5 x 2 (Number of Distractors x Attractor Width) within subjects analysis of variance. As expected, there was a significant effect for the Number of Distractors, $F(4,32)=5.1$, $p<0.01$ (Figure 15). Interestingly, in this condition, there was no main effect for the width of the attractor or an interaction. This might indicate that the additional *attraction* caused by a larger attractor was at least partially compensated for by the equally larger *distraction* caused by larger distractors.

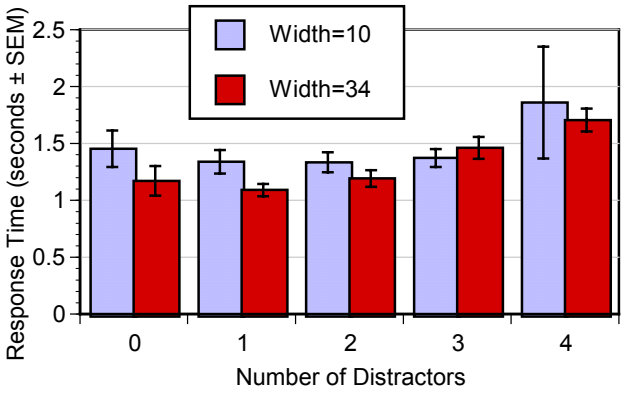

Figure 15: Task times *with* attractor at target and 0-4 distractors.

Target without attractor: We performed the same analysis for trials in which the target had no attractor, but there were still up to four distractors. This time the only significant effect was for Attractor Width (of the distractors). Unlike the case with an attractor at the target, there is no tradeoff here, so larger distractors impacted task time more. There was no main effect for the Number of Distractors or a significant interaction (Figure 16).

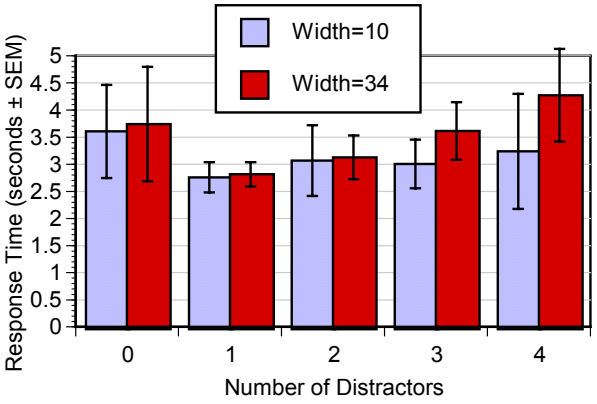

Figure 16: Task times *without* attractor at target and 0-5 distractors

These results provide some indication for deciding how many locations to provide with an attractor: successfully providing a target with an attractor saves users an average of 1.9 seconds per alignment interaction; the addition of four snapping locations not targeted by the user causes a penalty of less than 0.5 seconds.

USER STUDY 3: SNAP-AND-GO IN 2D

In our third study, we replicated the first two experiments for an alignment task in two dimensions. In this experiment each participant performed two tasks, the first being the 2D equivalent of the first user study, and the second being the 2D equivalent of the second user study. The study took participants about an hour to complete.

Task and interfaces

In both tasks, the participants' task was to align a square with two other squares as illustrated by Figure 17.

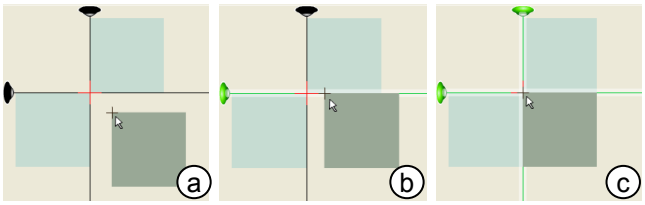

Figure 17: Task 1 of the 2D study. Participants aligned the dark square with the two lighter squares

Figure 18a illustrates the individual elements on the screen. Depending on condition, there were up to six line-shaped attractors. Three vertical attractors at the target location and 10 and 60 pixels right of it, three horizontal attractors at and below the target location. In the snap-and-go conditions attractors implemented the simplified snap-and-go behavior described in Figure 4d. Upon contact with the edge of the dragged rectangle attractor lines changed colors from black to green and displayed a white, 10-pixel wide halo. A light bulb at the end of each line informed participants about the current attractor strengths. The top left corner of the dragged rectangle was provided with a cross of 1-pixel lines to help visually verify alignment.

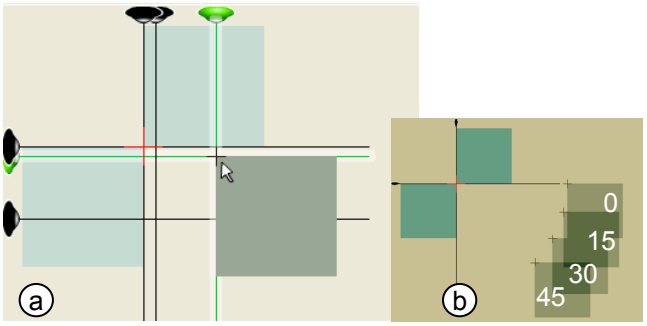

Figure 18: (a) Apparatus used in the 2D study. (b) The four start locations of the square.

Task 1: In correspondence to the first user study, only the two attractors aligned with the target were present (Figure 17). The design was within subjects 2 x 4 x 4 (Snapping Technique x Attractor Width x Approach Angle) with 8 repetitions for each cell. Distance to the target was 200 pixels and approach angles were 0, 15, 30, and 45 degrees, as shown in Figure 18b. Participants also performed 2 blocks of trials with snapping off at each distance.

Task 2: The design was within subjects 2 x 2 x 2 x 2 x 2 x 2 (Target Attractor Vertical x Target Attractor Horizontal x Distractor Vertical10 x Distractor Horizontal10 x Distrac-

tor Vertical60 x Distractor Horizontal60) with 4 repetitions per cell. This means that the target was enhanced with only the horizontal attractor, only the vertical attractor, both, or none. In addition, up to four distractors were located at the locations described above. Drag distance was always 200 pixels and all trials started at the 30 degree approach angle. To prevent sequence effects on Task 1 caused by unbalanced training, all participants performed Task 1 first.

Participants: Eleven volunteers (11 male) recruited internally participated. The average age was 31 years (std dev 8.0); all were right-handed.

Hypotheses corresponded to the 1D case. However, the cross widget used in the study causes more sideways drift than a combination of plus and bar widgets. We therefore expected distractors to have a slightly bigger impact.

Results

Task 1: snap-and-go vs. traditional snapping

As in previous experiments, we based our analyses on the median response time across repetitions for each participant for each cell. We performed a 2 x 4 x 4 (Snapping Technique x Attractor Width x Approach Angle) within subjects analysis of variance.

Traditional snapping was significantly faster than snap-and-go, $F(1,10)=13.1$, $p<0.01$ (Figure 19), and there was a significant effect for Attractor Width, $F(3,24)=97.6$, $p<<0.01$; performance improved as attractor width increased. There were no significant interactions or main effect for Approach Angle.

Performance vs. no snapping

We compared the most conservative case for the two snapping conditions (attractor size = 5) against no snapping for each distance. We performed a 3 x 4 (Snapping Technique x Target Angle) within subjects analysis of variance. There was a significant main effect for Snapping Technique, $F(2,20)=67.7$, $p<<0.01$. Planned comparisons of no snapping vs. traditional snapping and vs. snap-and-go were also significant, $F(1,10)=85.4$, $p<<0.001$ and $F(1,10)=60.0$, $p<<0.01$ respectively. See Figure 19. Again, there was no significant effect for Target Angle.

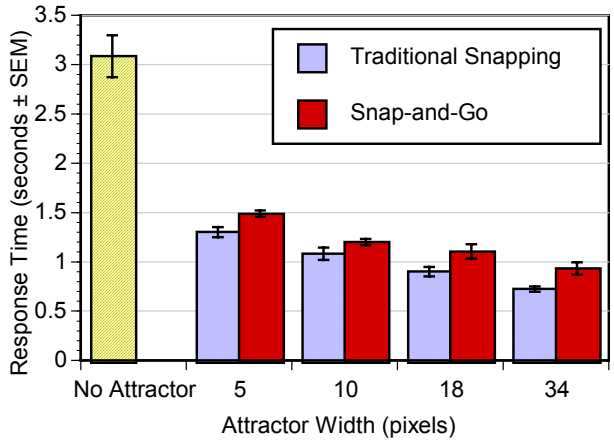

Figure 19: Task times in 2D by snapping method and attractor width.

Again, error rates were generally low. Differences in the error rates for the three different snapping conditions were not significant (No Snapping: 5.1%, Traditional Snapping: 3.7%, Snap-and-Go: 4.5%).

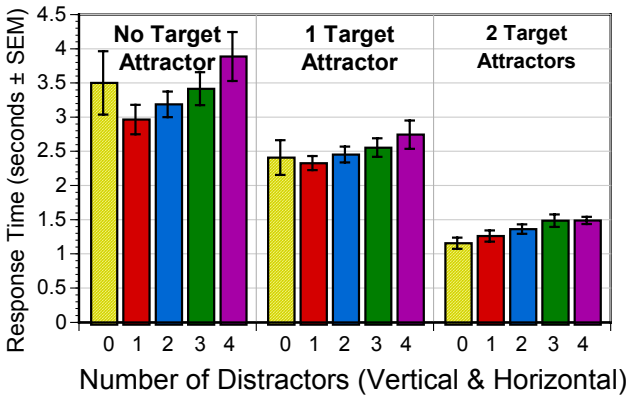

Figure 20: Response times for 2D distractor task with 0-4 distractors for each level of target attractor.

Task 2: 2D Distractor Task

As in experiment 2, the full factorial analysis was very difficult to interpret because of interaction effects. Initial results showed no significant effect for distractor position, so we simplified the analysis by combining the four binary Distractor variables into a single variable, Number of Distractors (0, 1, 2, 3, or 4). In addition, the two Target Attractor variables were reduced to 1, Number of Attractors (0 to 2).

We performed a 3 x 5 (Number of Target Attractors x Number of Distractors) within subjects analysis of variance. As expected, there was a significant effect for the Number of Distractors, $F(4,40)=3.7$, $p<0.01$, with movement time steadily increasing with the number of distractors (Figure 20).There was also a large main effect for Number of Target Attractors, $F(2,20)=69.8$, $p<<0.01$. As the number of target attractors move from 0 to 1 and then up to 2, performance improved markedly. There were no significant interactions.

DISCUSSION

In the three studies presented above, we covered snap-and-go in one and two dimensions. Across conditions, participants were significantly faster with snap-and-go than without snapping support. For the largest attractor width speed-ups were 138% in 1D and 231% in 2D. As expected, the additional motor space snap-and-go caused it to be slightly slower then traditional snapping, with differences of 3% in 1D and 14% in 2D. Snap-and-go turned out to be fairly robust against the presence of distractors.

Resulting design improvements

With traditional snapping, dragged objects are visibly warped when latching-on. During the first 1D study, some participants expressed how this visual cue helped them verify alignment. We therefore created a version of snap-and-go visuals that emulates warping using *anticipation behavior* [26, 9]: when latching on, the knob briefly over-

shoots and returns to the snap position; similarly, the knob first moves backwards when breaking free. We also created a version were the knob behaves as if dragged over little vertical barriers left and right of the snap location.

We also observed that switching from a traditional snapping condition to a snap-and-go condition caused some participants to converge particularly slowly towards the snap location—waiting for it to latch on. Since snap-and-go requires users to drag the knob *past* the apparent snap location, hesitant dragging continued all the way to the actual snap location, which affected task time. Figure 21 shows a redesign with corrected visual affordance. Here attractors are displayed *behind* the snap location. At the expense of introducing additional motion, this design also updates users about their location in motor space by moving the attractor against the mouse motion as the user passes it.

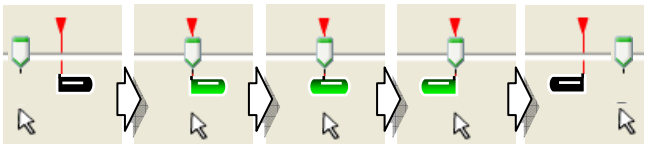

Figure 21: Redesigned attractor visual that always appears *behind* the target.

CONCLUSIONS

In this paper, we presented snap-and-go, an alignment technique that—unlike traditional snapping—does not require deactivation. While slightly slower than traditional snapping, the ability to omit the deactivation interface allows deploying snap-and-go in application areas where additional interface complexity would be prohibitive.

We made three main contributions. First, we demonstrated how manipulations of mouse gain can help users align objects. Second, we extended our technique to 2D by introducing the plus and the bar widgets that guide dragged objects to snap locations. And third, we presented three user studies evaluating snap-and-go in 1D and 2D in comparison with traditional snapping and no snapping.

As future work we plan to extend the snap-and-go concept to indirect pointing devices, such as pen and touch input.

ACKNOWLEDGMENTS

We would like to thank Cameron Etezadi, Gavin Jancke, Jordan Schwarz, and Phil Fawcett for their support of the snap-and-go project, as well as George Robertson, Maneesh Agrawala, Andy Wilson, and Noah Snavely for their comments on a draft of this paper.

REFERENCES

1. Accot, J., and Zhai, S. More than dotting the i's Foundations for crossing-based interfaces. In *Proc. CHI'02.* pp. 73–80.
2. Baudisch, P. The Cage: Efficient construction in 3D using a cubic adaptive grid. In *Proc. UIST'96,* pp. 171–172.
3. Beaudouin-Lafon, M. & Mackay, W. Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces. In *Proc. AVI'00,* p.102–109.
4. Beaudouin-Lafon, M. Novel Interaction Techniques for Overlapping Windows. In *Proc. UIST'02.* pp 153-154.
5. Bier, E. and Stone, M. Snap dragging. In *Proc. SIGGRAPH'86,* pp. 233–240.
6. Bier, E. Snap-dragging in three dimensions. In *Proc. 1990 Symposium on Interactive 3D Graphics,* pp. 193–204.
7. Blanch, R. Guiard, Y., Beaudouin-Lafon, M. Semantic Pointing: Improving Target Acquisition with Control-Display Ratio Adaptation. In *Proc. CHI'04*, pp. 519–526.
8. Borning, A. Defining constraints graphically. In *Proc. CHI 86*, pp. 137–143.
9. Chang, B.-W. and Ungar, D. Animation: From Cartoons to the user interface. In *Proc. UIST'93*, pp. 45–55.
10. Fitts, P., The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement, *Journal of Experimental Psychology,* v 47, June 1954, pp. 381–391.
11. Gleicher, M. and Witkin, A. Drawing with constraints. *The Visual Computer*, 11(1):39–51, 1994.
12. Guiard, Y., Blanch, R., and Beaudouin-Lafon, M. Object pointing: A complement to bitmap pointing in GUIs. In *Proc GI 2004,* pp. 9-16.
13. Hudson, S. Adaptive semantic snapping—a technique for semantic feedback at the lexical level. *Proc CHI'90,* pp. 65-70.
14. Igarashi, T., and Hughes, J.F. A Suggestive Interface for 3D Drawing . In *Proc. UIST'01*, pp.173-181.
15. Jul, S. This is a lot easier! Constrained movement speeds navigation. In *CHI'03 extended abstracts,* pp. 776 - 777.
16. Lécuyer, A., Burkhardt, J.-M., Etienne, L. Feeling Bumps and Holes without a Haptic Interface: the Perception of Pseudo-Haptic Textures. In *Proc. CHI 2004.* pp 239–247.
17. Lécuyer, A., Coquillart, S., and Kheddar, A. Pseudo-Haptic Feedback: Can Isometric Input Devices Simulate Force Feedback? In *Proc. IEEE VR2000*, pp.18–22.
18. Lieberman, H. editor. *Your Wish is My Command–Programming by Example*. Morgan Kaufmann Publishers, 2001.
19. MacKenzie, I.S. Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction 1992.* 7:91–139.
20. Masui, T. HyperSnapping. In *Proc. Symposia on Human-Centric Comp.—Languages and Environ. 2001*, pp. 188–194.
21. Michael McGuffin, Ravin Balakrishnan. Acquisition of Expanding Targets. In *Proc. CHI'02*, pp. 57-64.
22. Nelson. G. Juno, a constraint-based graphics system. *Computer Graphics*, 19(3):235–243, *Proc. SIGGRAPH'85.*
23. Raisamo, R. and Räihä, K.-J. A new direct manipulation technique for aligning objects in drawing programs. In *Proc. UIST'96,* pp. 157–164.
24. Sutherland, I. *Sketchpad: A Man Machine Graphical Communication System.* PhD thesis, MIT, 1963.
25. Swaminathan, K. and Sato, S. (1997) Interaction design for large displays. In *Interactions* 4(1):15 – 24.
26. Thomas, B.H. and P. Calder. Applying cartoon animation techniques to graphical user interfaces. *TOCHI* 8(3):198–222, September 2001.
27. Worden, A., Walker, N., Bharat, K and Hudson, S. Making Computers Easier for Older Adults to Use: Area Cursors and Sticky Icons. In *Proc. CHI '97*, pp. 266–271.