# Dirty Desktops: Using a Patina of Magnetic Mouse Dust To Make Common Interactor Targets Easier To Select

*Amy Hurst, Jennifer Mankoff, Anind K. Dey, and Scott E. Hudson*
Human Computer Interaction Institute, Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
{akhurst,jmankoff,anind,scott.hudson}@cs.cmu.edu

**ABSTRACT**
A common task in graphical user interfaces is controlling onscreen elements using a pointer. Current adaptive pointing techniques require applications to be built using accessibility libraries that reveal information about interactive targets, and most do not handle path/menu navigation. We present a *pseudo-haptic* technique that is OS and application independent, and can handle both dragging and clicking. We do this by associating a small force with each past click or drag. When a user frequently clicks in the same general area (*e.g.*, on a button), the patina of past clicks naturally creates a pseudo-haptic magnetic field with an effect similar to that of snapping or sticky icons. Our contribution is a bottom-up approach to make targets easier to select without requiring prior knowledge of them.

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

**General terms:** Design, Human Factors

**Keywords:** Adaptation, Snapping, Mouse, Pointer

**INTRODUCTION**
Selecting targets (such as menus, scrollbars, and buttons) is one of the most pervasive tasks in graphical user interfaces. As a result, even small problems with pointer control can be frustrating and slow user interaction. Adaptive techniques to help pointing can make computing more accessible to those with motor control difficulties and may even increase the efficiency of all users. As a result, adaptive techniques for improving pointing have been a focus of attention for decades [4]. Recently, pseudo-haptic techniques that rely on models of force fields, magnetism, or gravity, have been shown to increase interaction speed (*e.g.,* for target acquisition [1] and menu interactions [2]). In fact, in a real world setting where distracters pose problems for common target acquisition techniques, force-based target acquisition was shown to be more effective than simply making targets sticky by adjusting the control-display gain [1].

However, several challenges remain in creating these techniques. A primary challenge is identifying targets of interest. Additionally, much past work only deals with target acquisition, and not path navigation. Finally, pseudo-haptic techniques rely on hand-crafted force models.

Our contribution is a technique that addresses all three issues using information about past pointer events (clicks and drags) and window state (focus, size, and title). Using this information, we aggregate pointer clicks and drags into "magnetic mouse dust." Similar to past work on document navigation [8], this creates a patina over time that reflects where people click and drag most frequently. By associating a small attractive force with each pointer event, and adding them together, the pointer will naturally be drawn towards places that have been used frequently. An important aspect of our approach is that the effect on the pointer is small for any given point. It is only when the same region of the screen is systematically clicked on over and over again that this effect begins to build up. This means that occasional clicks in random places are not problematic.

While it may take time to build a useful patina of magnetic dust, this click and drag history can be shared among users. This allows a new user to get up to speed more quickly, or a user who has difficulty moving the pointer accurately (*e.g.*, due to a motor impairment) to leverage the accumulated pointer dust of people who have no accuracy problems. Since we associate magnetic dust with individual windows, it can easily be shared on a per-application basis.

Our tool differs from past work in its bottom up approach: past work starts by identifying targets, modeling them using attractive forces or snapping techniques, and selectively applying those techniques when the pointer approaches those targets. This results in reliance on unreliable accessibility libraries. Different toolkits and operating systems have different libraries, which may change over time, resulting in a need for costly implementation and ongoing

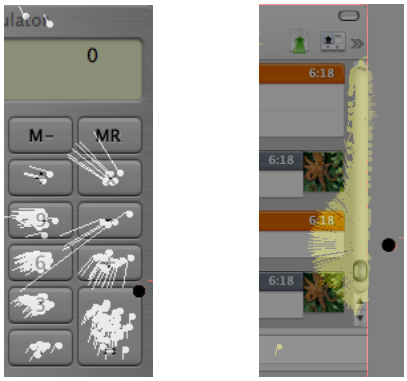


**Figure 1**: **(left)** An illustration of how clicks (white dots) collect on commonly used buttons in a calculator, creating a cumulative force (white lines) on the pointer (black dot). **(right)** The cumulative force (yellow lines) on the dragged pointer as it leaves the bounds of a scrollbar in an IM client.


*UIST'07*, October 7-10, 2007, Newport, Rhode Island, USA.

support. Also, not all interactors have been made accessible through those toolkits, including some used in popular applications such as the Color Chooser in Microsoft Office.

In contrast, we do not need to identify targets but can allow them to arise out of the dust created by past pointer events. This means that even when new and unexpected targets arrive, they will eventually be supported by our approach. Also, we use a physical model that works at the level of individual interactions, not individual interactors, and we do not define the components of this model but rather let them arise in a bottom up fashion. Finally, we do not need to know when the pointer approaches a target (since we do not know anything about targets), but rather make calculations each time we get a pointer event. The main advantage of our approach is its bottom-up nature, allowing it to support most any application with only mouse and window information and without depending on accessibility libraries. This makes it possible to deploy our application-independent approach with minimal OS specific changes.

The next section provides more details on our algorithm. Following that, our implementation section describes how our technique can support most common interactors, including both those that require simple target acquisition and those that require steering. We end with a small user study.

## THEORY

We begin by describing the basic algorithm for constructing additive magnetic forces from the pointer history. We use separate algorithms for target acquisition (*attracting the pointer to a place)* and steering (*keeping the pointer on a path)*. Implementation issues are discussed in the next section. In this section, we assume for simplicity that all pointer events occur in a fixed interactive context and that some magnetic dust already exists. Each click or drag (represented as a line) deposits new dust. This dust collects and exerts attractive forces on the pointer as it moves.

### Clicks

Mathematically, the simpler case is clicks, which we model as *point magnets*. Coulomb's law states that the magnitude of a force $F$ between two point charges ($q_p$ and $q_u$) is proportional to the charges (multiplied by the *electrostatic constant,* $k_c$) and inversely proportional to the square of the distance between the points ($r^2$) (See Equation 1).

$$\mathbf{F} = \frac{k_c q_p q_u}{r^2} \tag{1}$$

For our purposes, each pixel on the screen may hold a point charge. Each time a user clicks on a pixel, the charge in that pixel is increased, making it more magnetic. The user's pointer has a constant charge of 1 (causing $q_u$ to fall out of Equation 1), and we empirically determined that a value of 1 for $k_c$ works well. This gives the equation

$$\vec{\mathrm{F}}_{pixles} = \sum_{p=(0,0)}^{(r,r)} \frac{k_c q_p}{r^2} \hat{r} \tag{2}$$

where $\hat{r}$ is the unit vector pointing from the pointer to pixel $q_p$. As $r^2$ increases, the force due to $q_p$ becomes negligible (*i.e.*, for clicks far from the pointer). In addition to the forces exerted by pixels, the user is accelerating the pointer with a force $\vec{\mathbf{F}}_{user}$, calculated as the change in the pointer's velocity over time (Equation 3).

$$\vec{\mathbf{F}}_{user} = \frac{\Delta \vec{v}_m}{t} \tag{3}$$

This force needs to be added to $\vec{\mathbf{F}}_{pixels}$ to calculate the final force on the pointer (Equation 4).

$$\vec{\mathbf{F}}_{mouse} = \vec{\mathbf{F}}_{pixels} + \vec{\mathbf{F}}_{user} \tag{4}$$

### Lines

Lines occur when the user is controlling some of the more complex interactors (such as selecting a menu item or dragging the thumb of a scrollbar, as in Figure 1). In the case of a line, we simply record all of the pixels that make up the path of the pointer during the interaction. Each of those pixels becomes a point magnet, which attracts the pointer when it is close by. Typically, if the pointer is just off the line, it will be attracted to the line, while if it is on a (relatively) straight line, it will be in equilibrium.

Similar to clicks, if we add all the forces from multiple lines together, our system supports the right behavior. For example, when there are several parallel lines (such as those drawn when a scrollbar is used repeatedly), the pointer will be drawn to the central position among them. If two perpendicular lines cross, the forces due to both lines will approximately cancel each other out at the crossing point, making it easy for the user to switch directions.

## EXAMPLE

As an example, consider a user who frequently uses a calculator application. Over time, most of the clicks will fall on the center of buttons, as opposed to the spaces between buttons or areas closer to the borders (where button mishits or slips are common). Then, each of those buttons will develop a patina of magnetic dust and create an attractive field, as illustrated on the calculator in Figure 1 (note: the magnetic dust is shown only for illustrative purposes).

In Figure 1, there are 384 clicks on the calculator. These clicks create a cumulative force on the pointer in its current location (top right corner of the "=" button in Figure 1). Assuming the cursor is located at the black dot in Figure 1 and the user were to move the cursor towards the "=" button, our software would move the cursor an additional 3 pixels towards it. This creates the illusion that the pointer is being attracted to the pile of dust, and is similar to manipulating control-display gain [6].

Now take the example of the user scrolling in an IM client (Figure 1). When the user drags the thumb, she is creating line dust (672 drag points in our figure). We have found it confusing to "mix" dust – click dust should not affect the pointer when the user is trying to steer it, and line dust should not affect the pointer when the user is acquiring a target. For this reason, once the user presses the pointer and begins dragging it, we turn off the magnetic click dust, and turn on the magnetic line dust. This dust helps the user to move the pointer vertically down the scrollbar by attracting it back when it starts to move off the path – in the Figure 1 example, by 12 pixels towards the scrollbar.

## IMPLEMENTATION

We implemented our technique on a Macintosh using Java and C, with clicks and lines stored in an SQL database. Pointer events are logged by a C program based on code from [10] and written to the database. Each piece of magnetic dust is associated with a particular window, and only exerts a force on the pointer when that window has the focus. For lines, the associated window is the one that had the focus when the line started, and it typically does not change before the line ends. When a new pointer event arrives, it is sent from the C program to the Java program, which calculates the current force on the pointer, and uses the Robot class to move the pointer an additional amount based on that force.

For increased performance, we approximate the force from all points and lines greater than 300 pixels away from the pointer, as zero. For lines, this helps to select lines likely to be relevant based on the pointer's starting position (*e.g.*, all lines for a menu start in approximately the same region). We also use a threshold to ignore mouse motion above a certain velocity.

Next we explain how common interactors are supported with this approach, using our data set as an illustration. The original Macintosh guidelines include a fairly comprehensive set of interactors [3] and we supplemented this with Foley's description of interaction tasks [7]. It is important to note that while our ultimate goal is to enhance the experience of using these interactors, we must also "do no harm." Not every interactor is supported with our approach, but we argue that our technique does not make any interactors significantly *less* accessible.

### Click-Based Interactors

Interactors that respond to a single click include *Radiobuttons*, *Checkboxes*, and *Buttons*. *Lists* also fall into this class if they are static (*e.g.,* contain the same elements each time and are not scrollable). These interactors will naturally gather magnetic dust as they are used. They can be enhanced by the presence of attractive fields, and they are ideally suited to our approach.

### Menus

*Menu bars*, *Context Menus*, and so on, represent a class of interactors which can be enhanced by helping the user to move vertically (down a list of items) or horizontally (over to submenus). Lines are used to support this.

Menu interactions start when the user clicks on a menu (which is supported by the click dust). The user then moves the pointer down the menu and clicks on the item she wishes to select. To differentiate this from random clicking and motion, we assume that only right clicks (for context menus) and clicks at the top of the screen or window (for the Macintosh menu bar or Windows menus) represent the start of a menu interaction. When these clicks occur, we store them and all pointer motion until the resulting menu disappears (*e.g.*, drag ends or there is a second click).

### Scale Interactors

*Scrollbars* and *Sliders* are both well suited to our approach. As with menus, click dust can help the user to acquire the interactor. Also, for scrollbars in particular, click dust may help the user acquire up and down arrows. For those scale interactors that support clicking anywhere to modify the selection, click dust will be present all along the line, and will help to keep the pointer in bounds. In our experience, there will be a slight buildup of extra click dust at places such as the top of the scroll bar (the default location for the thumb). However, because of the long, narrow nature of scale interactors, this only has a detrimental effect if the user wishes to click close to, but not quite on, that location (other areas of the scrollbar are far enough away to not be affected by the attractive field).

As the user drags the pointer across a scale interactor, a patina of lines will build up that will also help to keep the pointer within the bounds of the interactor. Our technique does not help us find the location of the scrollbar thumb, but also does not make dragging any less accessible.

### Text Fields, Text Documents and Drawing Areas

*Text Fields*, *Text Documents* and *Drawing Areas* have more randomness associated with them, in the sense that the user may click or drag in a wider range of areas within them. As with scrollbars, attractive fields may develop in areas that are used frequently (*e.g.*, left side of text boxes), but this does not make it hard to reach uncommon places, just easier to get to common places. This is because those areas are far enough away to not be affected by the field.

### Dynamic Interactions

Our approach supports both window move and resize events. Since we store click events using the top left corner of the in-focus application window as a reference, the actual location of the window does not matter. We index dust maps according to window title and size, and store a new map for each combination. However, a limitation of our current approach is that it does not provide enhancement for dynamically changing window content; see future work.

## USER TESTS

To test our approach, we evaluated our prototype with 5 users (age=19-42, 5 male) who used computers at least 3 hours every day. We gathered a base data set of two days of continuous use from one author to test with our users. We then had each participant perform target acquisition and dragging tasks for approximately 45 minutes in an IM client and file browser with the adaptation on and then off.

Users liked the deployment of horizontal "gravity" while scrolling, and several mentioned that they often "lose" the cursor while scrolling, and felt that this gravity would be a nice way to prevent that from happening. All participants noticed the gravity adaptation while moving the mouse slowly, and said they did not notice the adaptation when moving quickly. Four of the participants felt it was easier to "miss" targets when the gravity adaptation was turned off. We saw a difference in users that were used to using mouse acceleration versus those who do not. Two of the users do not rely on mouse acceleration normally and were surprised to have their mouse move variably. However, they got used to the adaptation as they continued to use it.

## RELATED WORK

An excellent survey of past work in improving target acquisition summarizes approaches that manipulate the control-display gain to make targets "sticky" [4]. Physical models for this are rare outside of haptics literature (which most commonly uses gravity wells to cause the pointer to "stick" to a target, *e.g.,* [13]). However, recent work has looked at pseudo-haptic enhancements to traditional mice [1,2,6,12,14]. Force fields have been empirically shown to handle closely spaced targets better than stickiness for clicking [1] and to enhance menu use [2]. We review approaches to target acquisition and path following.

### Target Acquisition

Sutherland's thesis includes the first published example of a technique, now termed *snapping*, for making it easier to acquire targets [15]. He computes a "pseudo pen location" by moving the pen to an object of interest if it is within a certain radius of that object. Since then, a range of related techniques that manipulate the control-display gain have been developed and tested [4]. For example, Snap-and-go supports snapping to a target by increasing the target's size in motor space, thus creating the illusion that the pointer stops on the target before moving forward again [5]. In contrast, pseudo-haptic and haptic approaches support target acquisition using a force model. Most work in this area has focused on the design of haptic interactors, and thus assumes that the location, use, and shape of those interactors are known. Oakley *et al.,* lay out empirically based guidelines for the design of haptic user interfaces, such as making the maximum force strength configurable [13]. Target acquisition can also be supported by predicting the target of cursor motion (based on a known set of targets). Although target prediction is error prone, this research has matured sufficiently to be deployed successfully in an adaptations supporting target acquisition (*e.g.,* [11, 14]).

### Path Navigation

Path navigation has received far less attention than target acquisition in the literature. However, Jul's work on lodestones and leylines demonstrates that the combination of attractive targets and constrained motion to those targets significantly decreases time on task [9]. Additionally, recent work has shown the value of force fields for supporting menu navigation [2].

### Summary

The techniques presented in this section depend upon (1) prior identification of targets needing augmentation and (2) manual design of force fields or other warping mechanisms associated with those targets. In contrast, our work uses a bottom up approach in which past user data determines which targets are augmented and how, instead of relying on prior knowledge of target location.

## CONCLUSIONS AND FUTURE WORK

We presented a technique for enhancing the accessibility of pointer-based interactions by gathering historical data about pointer use. The resulting piles of magnetic dust naturally congregate in places the user might wish to put the cursor, creating attractive forces that support interaction.

In future work, we plan to explore hybrid techniques to handle dynamic layout changes in a window using computer vision and accessibility APIs. Dynamic layouts and interface changes could be supported by using the accessibility APIs to probe for information about on screen interactors. While simple and accurate, this solution is not portable and would not support all interactors. An alternative is to use computer vision to associate dust maps with screen regions that can be classified as highly similar. Points in these dust maps would be stored as positions relative to the visual element (and possibly scaled to its size).

Other future work includes exploring social techniques for gathering and labeling data.


## ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation grants EEC-0540865, IIS-0205644, IIS-0325351, and the first author's NSF Graduate Student Research Fellowship; IBM Research; and the Pennsylvania Infrastructure Technology Alliance.


## REFERENCES


1. Ahlström, D. Hitz, M. and Leitner, G. An evaluation of sticky and force enhanced targets in multi-target situations, *Proc. of Nordic HCI'06,* 58-67.
2. Ahlström, D. Modeling and improving selection in cascading pull-down menus using Fitts' law, the steering law and force fields. *Proc. of CHI'05*, 61-70.
3. Apple Computer, Inc. *Macintosh Human Interface Guidelines*, Addison-Wesley Professional, 1982.
4. Balakrishnan, R. "Beating" Fitts' law: virtual enhancements for pointer facilitation. *IJHCS* **61**(6):857-874, 2004.
5. Baudisch, P., Cutrell, E., Hinckley, K. and Eversole, A. Snap-and-go: Helping users align objects without the modality of traditional snapping. *Proc. of CHI'05,* 301-310.
6. Blanch, R., Guiard, Y. and Beaudouin-Lafon, M. Semantic pointing: improving target acquisition with control-display ratio adaptation. *Proc. of CHI'04*, 519-526.
7. Foley, J. D., Wallace, V. L. and Chan, P. The human factors of computer graphics interaction techniques. *IEEE Computer Graphics Applications,* **4**(11):13-48, 1984.
8. Hill, W. C., Hollan, J. D., Wroblewski, D. and McCandless, T. Edit wear and read wear. *Proc. of CHI '92*, 3-9.
9. Jul, S. "This is a *lot* easier!": Constrained movement speeds navigation. *Proc. CHI'03 Extended Abstracts,* 776-777.
10. Kukreja, U., Stevenson, W. E. and Ritter, F. E., RUI: Recording user input from interfaces under Windows and Mac OS X, Behavior Research Methods 06, 38 (4), 656-659.
11. Lane, D. M., Peres, S. C., Sándor, A. and Napier, A. H. A process for anticipating and executing icon selection in graphical user interfaces*, IJHCI* **19**(2):243-254.
12. Lecuyer, A., Burkhardt, J.M. and Etienne, L. (2004). Feeling bumps and holes without a haptic interface: the perception of pseudo-haptic textures. *Proc. of CHI'04*, 239-246.)
13. Oakley, I., Adams, A., Brewster, S. and Gray, P. Guidelines for the design of haptic widgets, *Proc. of the 16th BHCI Conference*, 195-211, 2002.
14. Rodgers, M. E., Mandryk, R. and Inkpen, K. Smart sticky widgets: Pseudo-haptic enhancements for multi-monitor displays. *Proc. of Smart Graphics'06,* 194-205.
15. Sutherland, I. E. Sketchpad: A man-machine graphical communication system. Doctoral Dissertation, *Massachusetts Institute of Technology,* 1963.