Part V.

User Interfaces

23. Introduction to User Interfaces

Interactive systems in computer science and engineering are normally considered technical objects. In contrast, the part of an interactive system that 'faces' the user is called a user interface. A user interface is the only part of a system that is used by real people doing real tasks in real settings. Therefore, what makes an interface good is ultimately determined by humans. Research on user interfaces is therefore, by necessity, a cross-disciplinary effort with contributions from computer science, electrical engineering, design, and psychology.

Innovations and insights into user interfaces take many forms. Engelbart [222] suggested that interactive systems and the user interfaces to them are really about augmenting the human intellect. That is, user interfaces should make people smarter. At a different level, Shneiderman [745] identified direct manipulation—the idea that you can directly operate on objects in user interfaces—as the success behind graphical user interfaces. As a final example, Weiser [860] suggested that the role of user interfaces is to disappear. A good user interface is one that no longer imposes itself on the user's attention.

The integrative theme in user interface design is that *a user interface should match the needs and capabilities of its users*. That is, on the one hand, they should be designed to respect the general understanding of humans as computer users, as laid out in Part II. From this understanding follows a set of objectives that are always relevant to consider in the design and evaluation of user interfaces. On the other hand, every user interface is unique when it comes to specific users and their contexts. Knowledge about users and their contexts usually comes from empirical research (see Part III). However, user interfaces can also be understood more abstractly through theories and models, which are discussed in Part IV. User interfaces enables interaction. That way, a good user interface draws on input from these three sources.

In the rest of this chapter, we first define user interfaces and give some examples of user interface types. Each of them is discussed in depth in subsequent chapters. At the core of all work on user interfaces is a thorough understanding of what an interface is, what the key design choices are, their associated trade-offs, and how to approach open decisions systematically. The last sections of this chapter discuss these choices and trade-offs.

23.1. Definition and Elements

A user interface can be broken down into four constitutive elements, according to the DIRA model [67].

Devices To be effective, an interactive system must allow the user to provide information and control it effectively, efficiently and safely. Input devices allow the user to provide input to the system in a variety of ways. Examples of input devices include mice, keyboards, touchscreens, microphones, and eye trackers. In addition to the user providing information to the system, the system needs to provide information to the user. Output devices, or often just 'displays', allow the system to provide information and feedback to the user. Laptop displays, audio, rumble, and force-feedback are examples of output devices.

Interaction techniques Input devices provide input from the user, but do not necessarily interpret these inputs in a manner that helps operate an interactive system. Interaction techniques translate input from input devices and possibly context sensors into solutions to basic interactive operations, such as moving, selecting, pointing, or navigating. The gain function, which maps mouse movement to cursor movement, is an example of an interaction technique (see Chapter 26).

Representations Representations determine how data, events, objects, and actions appear to the user. Representations may be text, sounds, taste, or any other form of data in an interactive system that is shown to users through a display. The design of visual icons (e.g., the thrash can) is an example of representations.

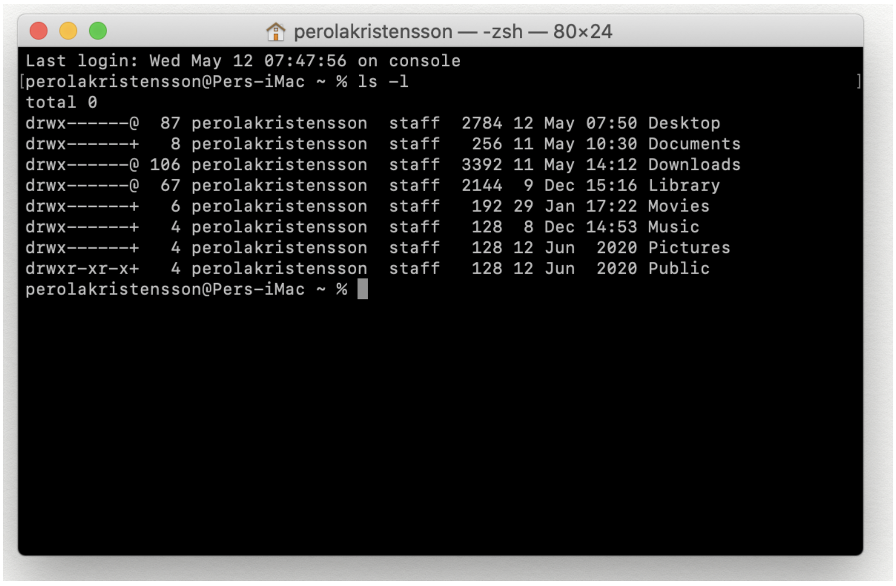
Assemblies describes the broader set of principles that organize representations. The assemblies define how the representations are organized and put together in the UI both spatially and temporally. For instance, in a web browser, pages are linked in particular ways, pages update their representations in particular ways, and layout information in particular ways. Those ways are the essence of assemblies.

Together, these elements form a user interface. Note that this definition does not mention processing. The internal mechanisms of how the system processes information about the user are not part of the definition. Instead, the user interface is the part of the technical system that is perceivable and actionable from a user's perspective. The user interface and its broader technical operating context are jointly called *an interactive system*.

In this part, devices are discussed in the chapters on input devices (Chapter 24) and displays (Chapter 25). Chapter 26 discusses interaction techniques and Chapter 27 discusses some central issues in representation. The question of organization for individual interface types is discussed in the chapters that cover such types; next, we discuss the main types of user interface.

23.2. Interaction Styles

User interfaces can take many styles and forms, depending on the tasks they need to support users with. To understand the fundamental ideas in user interfaces, it is useful to be aware of a concept known as *interaction style* in HCI. An interaction style is a particular genre of interaction that specifies how users interact with computer systems. An interaction style is a combination of particular instances of the four elements of user interfaces that recur in the literature or in practice. Each has particular characteristics, advantages, and disadvantages. Next, we briefly discuss five styles of interaction. Note



```
Last login: Wed May 12 07:47:56 on console
perolakristensson@Pers-iMac ~ % ls -l
total 0
drwx------@  87 perolakristensson  staff  2784 12 May 07:50 Desktop
drwx------+   8 perolakristensson  staff   256 11 May 10:30 Documents
drwx------@ 106 perolakristensson  staff  3392 11 May 14:12 Downloads
drwx------@  67 perolakristensson  staff  2144  9 Dec 15:16 Library
drwx------+   6 perolakristensson  staff   192 29 Jan 17:22 Movies
drwx------+   4 perolakristensson  staff   128  8 Dec 14:53 Music
drwx------+   4 perolakristensson  staff   128 12 Jun  2020 Pictures
drwxr-xr-x+   4 perolakristensson  staff   128 12 Jun  2020 Public
perolakristensson@Pers-iMac ~ %
```

Figure 23.1.: An example of command entry at a command prompt in a terminal window. The command `ls -l` lists all files and directories in the current directory.

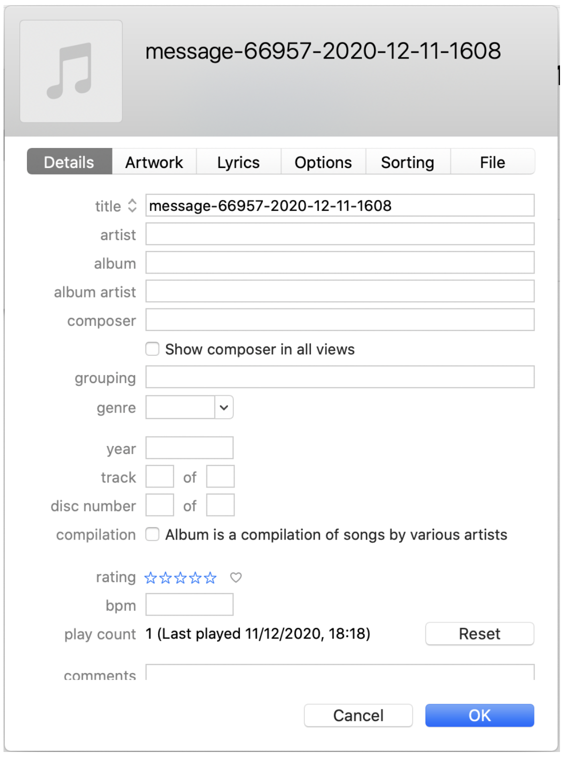
that an interactive system user interface may mix several of these interaction styles. The combination of the interaction styles of menus and forms is common on the World Wide Web.

23.2.1. Command-line interfaces

In a command-line interface, the user enters commands and waits for the interactive system to respond. Classically, this happens in a terminal that offers nothing but a textual prompt for the user. The response from the system is also typically just text with which the user cannot interact directly, except for issuing another command.

The best example of command-line interfaces came from the first user interfaces to computers, where commands are typed into a terminal (Figure 23.1). Nowadays, many development environments include an integrated command prompt, as do graphical user interfaces (as in the terminal in MacOS and Command window in Windows).

Command-line user interfaces are now typically provided to support expert users with tasks that are impossible, difficult, or inefficient to carry out in a point-and-click interface. This is powered by their ability to abstract over objects (for instance, using a wildcard operator) and be customizable by options and multiple parameters. The drawback is that the user needs to recall commands. Also, command-line user interfaces can have syntactic errors (e.g., misspellings) and do not prevent meaningless operations (e.g., printing a folder).



message-66957-2020-12-11-1608

title	message-66957-2020-12-11-1608
artist	
album	
album artist	
composer	
	☐ Show composer in all views
grouping	
genre	
year	
track	of
disc number	of
compilation	☐ Album is a compilation of songs by various artists
rating	☆☆☆☆☆ ♡
bpm	
play count	1 (Last played 11/12/2020, 18:18)
comments	

Figure 23.2.: An example of form in a GUI.

23.2.2. Forms

The use of *forms* is considered another style of interaction. A form provides support for data entry and first arrived in text terminal systems. They consist of one or more widgets that the user can input through (for instance, text boxes or checkboxes). Figure 23.2 shows an example of a form. Many of the conventions of the forms are still in place today, such as the use of the `Tab` key to move between different forms.

FOrms are useful for supporting data entry, as they provide a highly structured and easy-to-understand interface to collect the required information from users. They help understand what to input and in what order. However, forms are highly limited to data entry tasks, and even then, their efficacy and utility for such tasks depend on the ability to efficiently encode all data entry concerns into a form that is easy to understand and process.

23.2.3. Menus

Another central interaction style is *menu* interaction. A menu presents a user with a set of options of which one can be selected; the selection may trigger a command or may open another part of the menu. Similarly to forms, menus can be the only interaction style in a UI, or be part of other interaction styles. A traditional GUI provides linear drop-down menus that let a user explore a menu hierarchy (see Figure 23.3). Menus do

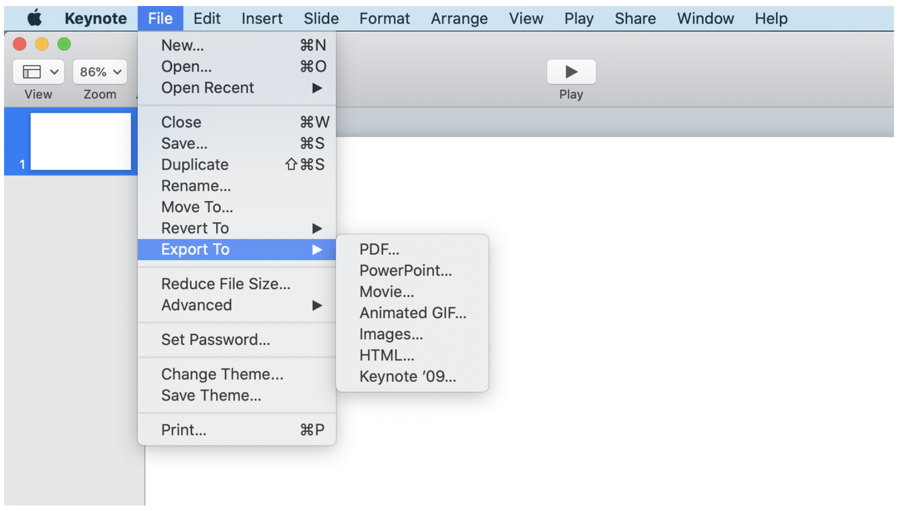


Figure 23.3.: An example of linear pull-down menus in a presentation software GUI.

not necessarily have a pull-down style. For example, another possible design is to reveal all menu items directly.

Menus have several advantages. First, they support exploration. Unfamiliar users searching for a particular command can explore the menu structure to find it. Second, the menus can help both novice, intermediate, and experienced users. Novice users can spend time exploring the menu structure and gain familiarity with the availability of commands. Third, pull-down menus allow more commands to be made available compared to showing all commands to the user on the screen. Since top-level menus can be decomposed into both menu items and submenus, pull-down menus also scale and allow natural groupings of commands. A major disadvantage is that the scalability of menus is limited in practice. Even though menu hierarchies can in theory be massive, in practice they become cumbersome to navigate and difficult to explore. Another disadvantage is that frequently used commands may be easier to access for novice and intermediate users if they are present in a toolbar instead of forcing users to navigate or explore menus.

23.2.4. Graphical User Interfaces

Graphical user interfaces (GUIs) allow users to interact directly with objects in the user interface. This ability is known as *direct manipulation*. An example of such direct manipulation is when the user deletes a file by moving the mouse cursor over the file, selecting it, and subsequently dragging it to an icon representing a bin. The graphical user interface has been the dominant interface paradigm since the Xerox Star in 1981 (see Figure 1.1). A central idea associated with GUIs is direct manipulation, as discussed at the beginning of this chapter.

The benefits of GUIs are that they support novices in seeing the available objects and actions, and that direct manipulation makes it possible to explore the user interfaces because the user can undo operations. The drawbacks are that it is not always effective to present objects and actions visually and that the ceiling of performance may be lower than, for instance, for command-line interfaces. We discuss these benefits and drawbacks in more detail in a chapter devoted to GUIs (Chapter 28).

23.2.5. Reality-based interaction

Reality-based interaction is an umbrella term that covers the styles of interaction that have been developed since the GUI [374]. Such interfaces attempt to draw on our experience with the real world to develop new forms of user interfaces. This style covers a broad range of diverse interfaces.

To preview the advantages and disadvantages of reality-based interaction, let us consider some examples. In *mobile user interfaces*, the user is not restricted to use the interactive system while stationary, typically at their desk in an office. Calculator watches and personal digital assistants (PDAs) are early examples of devices that allowed users to interact with computer systems off the desktop. Recent examples include touchscreen mobile phones, tablets, smartwatches, and fitness bands. Mobile user interfaces have their set of challenges, most notably the small form factor.

Another example is mixed reality (XR) and augmented reality (AR), which allow users to interact with virtual objects registered in the physical world surrounding the user. An example of an augmented reality interface is a user moving their phone around and the display revealing virtual objects that appear located in the 3D background of the actual physical room. Another example is a user wearing an optical see-through head-mounted display that allows the system to display virtual objects that appear located in the physical surroundings of the user's physical location. As the user moves their head around the scene, any virtual objects will appear as they are seamlessly part of the physical location.

More information is given in Chapter 29, including other styles of reality-based interaction.

23.3. Design Objectives

HCI researchers and practitioners often focus on the design objectives of user interfaces. What are the things that are important in designing such interfaces? If we understand those objectives, we can consider them in thinking about user interfaces, in designing them, and in evaluating them. In general, they come from three sources.

- Understanding of people. User interfaces must fit the people who use them, including how they perceive, how their memory works, how they learn, and how they collaborate. These have been described in Part II. For instance, the manual control of a user interface needs to be learned and remembered. Thus, learnability of an interface is important.

- Our theories of interaction. User interfaces enable interaction with interactive systems. Thus, the views presented in part on Interaction (Part IV) give ideas for what to aim for in a user interface. For instance, the chapter on information and control (Chapter 17) discusses input rates; maximizing these is a key design objective for user interfaces.

- Attributes and structural qualities of the user interface. These objectives are close to the structure of the user interface and the way widgets are designed. For instance, consistency is an often pursued design objective for user interfaces. The idea of this objective is that consistent user interfaces are typically easier for people to operate (but not always, as we shall discuss).

Note that a discussion of design objectives for user interfaces brackets some issues around utility (see Chapter 19) and the fit of an information system to practice (Chapter 22). This is because these issues are not primarily related to the user interface as such, but rather to its fit to people, activities, and context (for a refresh, see Section III). With that in mind, we next discuss some objectives that recur in discussions of user interfaces. Throughout this part, we discuss other such criteria.

23.3.1. Supporting Novice, Intermediate, and Expert Performance

A central objective of a user interface is to allow users to learn how to use it, become proficient, and use it to enable to achieve their goals. However, there are many aspects of learning and proficiency in a user interface. Figure 23.4 shows an overview of these aspects. First, a central concern is that user interfaces are *easy to learn*. It needs to be easy to walk up to a system and figure out how to operate it.

Second, another central concern is that users can become *proficient* with the user interface. This means that they should be able to seamlessly transition from novices, to intermediate users and experts. This requires the user interface to support users in gradually improving their performance to ultimately reach high levels of performance.

Third, some user interfaces may focus on the ability to achieve *optimal performance*. This may require extensive training and practice over time. For example, studies of computer gamers show that *StarCraft 2* experts use two commands per second [899] and experts in *Tetris* can manage four pieces per second [see 471, for an introduction to expertise in Tetris].

The focus on initial performance, extended learnability, and ultimate performance raises several objectives for user interfaces. These objectives have been articulated as criteria for evaluating interfaces [865], principles for designing user interfaces [576], models for transitioning from novice to expert in user-interface performance [157], and surveys of how learnability is conceptualized and measured [292]. Next, we go through the interface *features and strategies* identified in those papers; general insights about learning are covered in Chapter 5.

Initial Performance To support initial performance, it has been suggested that user interfaces should support exploratory learning [680, 865]. In exploratory learning, users investigate the system on their own initiative, testing features and strategies to solve real

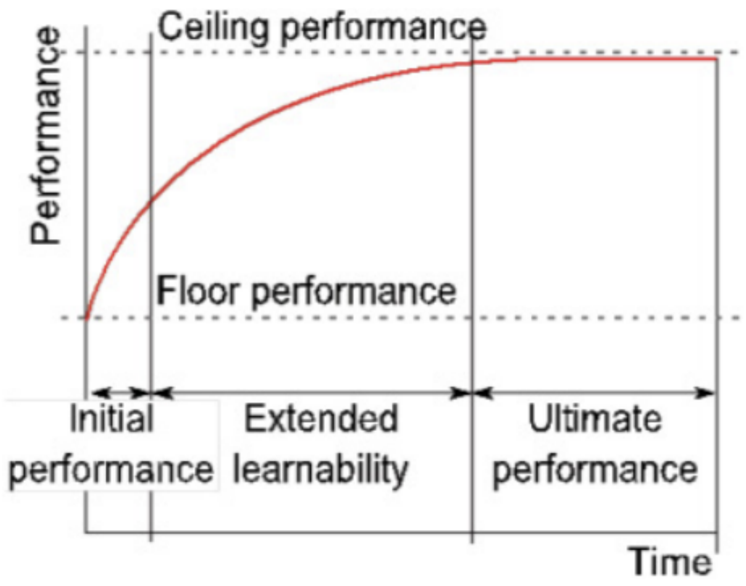


Figure 23.4.: A simple model of performance; on the Y-axis three phases of learning [157, p.12].

tasks or to learn about the system. User interfaces may support exploratory learning in many ways.

One strategy is to allow for the reversal of actions (for example, by making undo functionality available). This encourages exploration by allowing users to have an easy method to reverse any unintended changes in the state of the system.

Another strategy is to make actions and the status of the system visible, as discussed earlier in the principle of visibility. This allows users to recognize possible actions instead of having to recall command names or particular gestures. Making relevant aspects of the status of the system visible serves to reassure users that they are on the right path.

A third strategy to support the initial performance is to *simplify* the interface as users begin to learn it and only gradually add new layers of functionality and complexity. The idea of training wheels for bicycles is a everyday example of this; Paper Example 23.3.1 shows an example of this idea applied in HCI.

Paper Example 23.3.1 : Attaching training wheels to a graphical user interface

The idea of gradually adding new functionally as users become more proficient was introduced to the HCI literature as *training wheels* [132]. This concept identifies typical errors that novices make with an interface. It then prevents these errors by blocking certain states of the user interface or disallowing certain complex commands. In an experiment with a word processor, the concept of a training wheel resulted in faster and better learning of the interface. The key issue with simplifying user interfaces is when and how to transition to more complex interfaces. An approach to doing so is to dynamically reveal the features of the interface.

The idea here is to show the features when the user may need them, and to hide them otherwise. One commonly implemented tactic is to use context-dependent toolbar panels that dynamically change to reflect the relevant commands for the user's current selection. For example, in a word processor, a context-dependent toolbar may show commands relevant for editing a table when the user has selected a table and commands for editing a figure when the user has selected a figure.

In an experiment with a word processor, the concept of training wheels resulted in faster and better learning of the interface.

Extended Learnabilty For extended learnability, other objectives are needed. The goal here is to help users get closer to performing near their ceiling performance with the user interface. Interface features that are known to support this include the following:

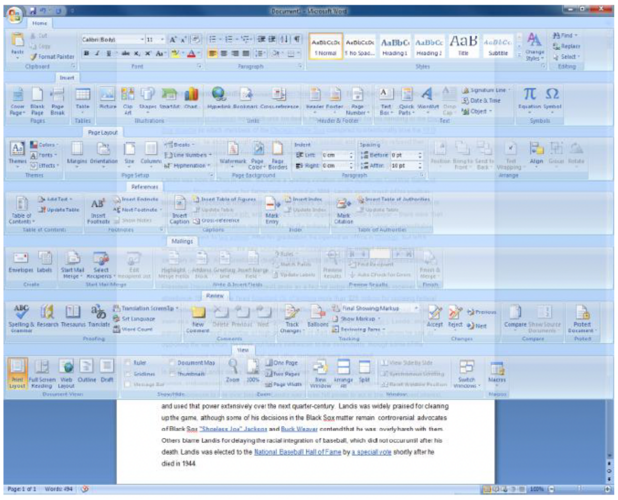
1. Shortcuts, also known as hot keys, which provide the user with an alternative method for accessing commands using a keyboard combination, such as `Ctrl+C`, instead of traversing a menu structure.
2. Support for habit formation and automaticity. Raskin [669] has argued that modeless interfaces are helpful in supporting the formation of habits. In a modeless interface, all operations are available all the time; in a mode, some part of the user interface works differently (e.g., the caps key).
3. Battling satisficing: ways of rehearsing and integrating extended learnability in software. For instance, the user interface ExposeHK [499] helped its users discover and rehearse keyboard shortcuts.

Paper Example 23.3.2 shows another example of improving the performance ceiling.

Paper Example 23.3.2 : Improving Extended Learnability with CommandMaps

User interface design often needs to consider how much information to show; Scarr et al. [716] coupled this discussion to improving intermediate learnability in a surprising way.

In a typical hierarchical structure, one needs to click icons to open a small menu with further options. In contrast, the experimental interface CommandMaps relies on the user's spatial memory to flatten a hierarchical structure (essentially a menu UI) to improve the ceiling of performance for experts. As shown below , CommandMaps simply immediately expands the hierarchy of icons. This uses a lot of space on the display but ensures that each icon appears in the same location. This means that extended learnability may be enhanced because experts can perform much faster, while novices can still use the interface. Scarr et al. [716] used an empirical study to show that this was indeed the case.



and used that power extensively over the next quarter-century. Landis was widely praised for cleaning up the game, although some of his decisions in the Black Sox matter remain controversial: advocates of Black Sox "Shoeless Joe" Jackson and Buck Weaver contend that he was overly harsh with them. Others blame Landis for delaying the racial integration of baseball, which did not occur until after his death. Landis was elected to the National Baseball Hall of Fame by a special vote shortly after he died in 1944.

Ultimate performance: Some user interfaces may focus on the ability to achieve optimal or peak performance in the long term. What is the fastest way to input text into a computer that is possible to design? This is a difficult goal to reason about, but theoretical limits to performance derived from models of performance are once a source of input. We can use such limits to remove barriers to optimal performance.

Another way to reason about ultimate performance is to draw on models from the chapter on information and control (Chapter 17). Such models can help to reason about the maximal throughput of an input device, for instance. Finally, some people have taken inspiration from science fiction to imagine what superpowers its users would want and then reason about how to realize them in user interfaces [874]. For instance, we could imagine user interfaces that improve our ability to attend to information in the environment, such as being able to see things that other people miss (like Sherlock Holmes) or cannot detect at all (like Spiderman).

23.3.2. Usability and Accessibility

According to the chapter on tool use, a user interface is ultimately a means to an end: Users interact with user interfaces to achieve goals that are external to the interface itself (see Chapter 19). Therefore, user interfaces should *support* help users achieve their goals and, conversely, avoid preventing progress toward those goals. In particular, the concepts of usability and accessibility may work directly as design objectives for user interfaces. Recall that *usability* concerns whether users can achieve their goals effectively, efficiently, and with satisfaction. *Accessibility* refers to a goal of having equivalent levels of usability between user groups. The reader may consult Chapter 19 to see details of these goals.

But how can we turn these views of interaction into more concrete objectives for thinking about user interfaces? One attempt to do so is to articulate guidelines or criteria that good interfaces should adhere to. We will discuss such guidelines in detail in the part on evaluation (Part VIII), but such guidelines may also be used to think about user interfaces in a more general way. In heuristic evaluation [574], a method to be discussed in Chapter 41, we might see the following heuristics for how to ensure that an user interface is usable.

- Help users recognize, diagnose, and recover from errors. Provide error messages that are understandable by users and offer a clear solution path to rectify the problem.
- Error prevention. The user interface should be designed to prevent errors, for example, by displaying a warning and requiring user confirmation before a non-reversible action is triggered, such as deleting a file.

To make user interfaces efficient, we may consider the following. Note that we use the term operation here to mean accomplishing things like selecting, navigating, entering text, and so on. This is different from tasks in the sense of what the user might want to achieve with a system in a matter of hours or days. But for operations, users generally expect to achieve their goals as quickly and accurately as possible.

- Flexibility and efficiency of use. Since users inevitably vary in their proficiency of a user interface, it is often effective to provide interface features that tailor to different users. A simple example is providing keyboard shortcuts for menu items and toolbar buttons that allow an expert user immediate access to these functions, while a novice user can still use direct manipulation to easily locate them (albeit at a slower pace). Another strategy is to allow users to customize the user interface or have the user interface adapt to the user.

In short, we have some ideas of what a usable interface normally looks like (as expressed in the guidelines above), and we may use those ideas in the design and evaluation of user interfaces.

23.3.3. Structure and attributes of the user interface

User interfaces are typically evaluated at a ’lower level’ than full interactive systems. Thus, their design objectives can be much more closely related to the four elements of user interfaces than the goals discussed above. The following list is incomplete but represents the main design objectives of user interfaces.

Explorability

means that a user interface supports exploratory learning of the functionality offered [680, 865]. In exploratory learning, users investigate the system on their own initiative, testing features and strategies to solve real tasks or to learn about the system. User interfaces may support exploratory learning in many ways. One is allowing reversal of actions (e.g., through undo). Another strategy to support initial performance is to *simplify* the interface as users begin to learn it, and only gradually add new layers of functionality and complexity.

Discoverability

is about the effort it takes to find the possible actions in a user interface; it is closely related to the previous point. Norman [580] considered it the question of "Is it possible to even figure out what actions are possible and where and how to perform them?".

One strategy is to actions visible. This allows users to recognize possible actions rather than having to recall command names or particular gestures. In heuristic evaluation, the idea is to support discoverability through through the visibility of system status. The current state of the system should be visible to the user. A simple example is a progress bar that indicates the progress of a long-term operation (such as downloading a large file).

Consistency

means that the representations (see above) and ways of interacting are similar across the user interface or with other (reference) interfaces. If the print command is represented with an icon showing a particular printer in one part of the interface, the same icon should be used in the other. Similarly, for interaction techniques. If one has to right-click an object to invoke a particular command, one should use the same right-click for other commands. Consistency might be followed across user interfaces (the so-called external consistency). The rationale is that external consistency helps users transfer skills from other interfaces.

There are several techniques to ensure consistency. Adhering to style guides—such as Apple's Human Interface Guidelines—ensures consistency across applications on a platform. Tools have also been developed that can help check the consistency of the bottom placement and labels [495]. However, note that the notion of consistency is not clear-cut. For instance, Grudin [294] provided several examples of interface-design decisions where consistency was not attractive.

23.3.4. Tradeoffs among objectives

When discussing the design objectives for user interfaces, it should be realized that these objectives sometimes trade off. This introduces a central challenge of user interfaces:

you cannot have your cake and it. Consistency is a classic example. As mentioned above, consistency is essentially about ensuring familiarity. However, it compromises novelty, another design objective. Some *trade-offs* central to the user interface are listed in Table 23.1.

Most choices in user interface design involve trade-offs. Consider a simple binary decision, for example, whether to include a captcha on a login page. Captchas uses a simple question to confirm that the user is a human and not a bot. Including a captcha may decrease the probability of system faults due to attacks by bots. On the other hand, it lowers user performance: it simply takes longer to do the login. *Not* including a captcha, by contrast, makes logging in faster. This is what users would prefer. On the other hand, system faults, which can be catastrophic, may increase. Thus, the decision to include a captcha requires resolving a usability–vulnerability trade-off.

HCI is not alone with tradeoffs. Tradeoffs are inherent to all multi-objective problems in design and engineering (see Section **??**). *Pareto frontier* is a technical concept that generalizes the concept of tradeoffs. It is used to describe the set of 'equally optimal' designs. Pareto optimal designs are optimal designs that strike different compromises among the objectives. How are such tradeoffs solved in practice? Practical settings may include consultation of design guidelines (Section 2), analytical methods (Section **??**), or theories on interaction (Part IV). They may also be resolved through empirical evaluations.

23.4. Design Space Analysis

User interfaces are complex. A successful user interface is a combination of input and output devices that are assembled by well-designed interaction techniques and representations that are appropriate for the tasks users are expected to carry out. A decision about a user interface should start by considering it in the context of the totality of decisions. How do you grasp a user interface without getting lost in the details?

Design space analysis refers to a semi-formal representation of the constituents of user interfaces [494]. Design space analysis is an essential thinking tool. It avoids getting lost in detail by enumerating the key choices and comparing them, and can thereby help both technical problem-solving and higher-level reflection.

Table 23.2 gives an overview of three such techniques. They differ in their focus and unit of analysis. *Morphological analysis* is best suited for understanding a user interface as a transducer: something that transforms a user's actions into changes in the computer's state. It offers an abstraction of the user interface that is not tied to the particular technology on which the interface is implemented. *Joint system analysis*, which also comes from engineering, adds humans to the picture. The properties of user interfaces are described as parameters, some of which are controllable by the designer (color of a button), but some of which are not (e.g. user experience). The benefit of this approach is that it can help to see the connection between technical considerations and end-user related objectives. Finally, QOC (Questions, Options, and Criteria) is an analysis method that aims not only to enumerate, but also to help decisions. In addition to asking to

Trade-off	**Explanation**	**Example**
Speed vs. accuracy	Improvements in task completion time compromise accuracy	When we make pointing devices faster we often affect their accuracy, and vice versa (see Section 24).
Recognition vs. recall	Relying on recognition memory makes learning faster but curbs maximum achievable performance, while reliance on recall slows down learning but permits high performance over time	User interfaces such as command language interfaces versus graphical user interfaces.
Familiarity vs. novelty	Following conventions and standards improves learnability but compromises felt novelty	Novel services often want to stand out by UI design, but users often want familiarity.
Expert vs. novice use	Trying to cater for expert users may make the UI hard to use for novices and thereby detract from user adoption	Professional photo-editing software that offers thousands of areas of functionality may be very hard to use for novices.
Mismatch between those who do the work and those who reap the benefit	User interfaces may shovel low-level tasks like information filling to one user group while another benefits from this	User interfaces for corporate information systems may be useful for management but not for end-users.

Table 23.1.: Central trade-offs in user interface design.

enumerate key decisions, it enumerates possible alternatives (options) and criteria (what is desirable).

Design spaces may be put to many uses. Haeuslschmid et al. [302] developed a design space for interactive windshield displays in vehicles by drawing together and analyzing existing literature. Markussen et al. [504] were interested in developing a selection-based text entry methods to be used in midair. They established a design space for such methods, by enumerating the central design questions, the possible user interface options, and the criteria to choose among them.

Method	**Focus**	**Unit of analysis**
Morphological analysis	Technology	Technical variables that constitute an interface
Joint system analysis	Joint system that includes the human	Parameters that describe the properties of the joint human–computer interface
Questions, Options, and Criteria (QOC)	Design decisions and their rationale	Questions in design that are mapped to options and objectives (criteria) for passing

Table 23.2.: Three forms of design space analysis that can be used to understand user interfaces.

23.4.1. Morphological Analysis

Morphological analysis refers to the idea of representing user interfaces as points in a parametrically described design space. The roots of this idea can be traced to Herbert Simon and Francis Zwicky. Zwicky, who worked at the Jet Propulsion Lab (JPL) on rocket engineering, proposed attacking very complex engineering problems by what he dubbed morphological analysis, which entails a breakdown of decisions to form a discrete solution space. Simon proposed that many complex-looking problem-solving tasks, such as in chess, are essentially search problems in a tree. At any given moment, you have possible actions. Depending on what you choose, another state opens with another set of actions. Following this line of thinking, user interface design is simply a set (Zwicky) or sequence (Simon) of decisions.

Card et al. [130] introduced these ideas to HCI. For example, an input device is a transducer that associates sensed properties of the physical world with parameters of an application. Formally, then, its morphological design space is a tuple:

$$< M, In, S, R, Out, W >$$

where

- M is a manipulation operator (physical action by the user that produces inputs)

- In is the input domain (range of possible values produced by the input manipulation operator)
- S is the current state of the device
- R is a resolution function that maps from the input domain set to the output domain set
- Out is the output domain set
- W is a general-purpose set of device properties that describe additional aspects of how a device works.

Most components and widgets of the user interface can be analyzed in this way. For example, consider a binary toggle that is operated by touching the finger within its bounding box, a touchscreen display. Currently, it is set to the leftmost position and is off. If we consider this through a morphological analysis, we get the following.

- M is a x and y coordinates of a touchpoint registered within the bounding box of the toggle
- In is a binary event: a touch occurred (x and y coordinates are ignored)
- S is 0 (off)
- Out is 0 (on) or 200 (off); if 'on', the toggle indicator turns green; if 'off', it turns gray

This view is expanded in Section **??** where we discuss optimization as an approach to user interface engineering.

23.4.2. Joint System Analysis

A user interface can also be thought of as a *joint system* that links users with computing technology. A menu interface consists of not only the decisions that tell which menu command goes where, but also the key capabilities the user uses to operate the menu, such as perception, cognition, motor control, and so on. From this perspective, the design of a user interface can be thought of as an analysis of controllable and uncontrollable parameters that affect joint outcomes.

A *controllable parameter* is a design parameter that the designer can directly influence, such as the colors, size, animation, and behavior involved in the design of a push button. Such parameters can therefore also be tuned by the designer or optimized for some criteria of interest, such as minimizing task completion times or minimizing errors.

Critical distance is the maximum change in the value of a controllable parameter that causes an undesirable drop in some design objective. For example, consider a fully packed graphical layout that uses all the display space. In this case, adding just one more pane to the layout might devastate the readability of the rest of the layout. In this case, the critical distance of that parameter was 1.

In contrast, an *uncontrollable parameter* is a parameter that the designer cannot directly influence, but that is nevertheless relevant to users achieving their goals in a user interface. An example of an uncontrollable parameter is the system delay in a web browser attempting to load information from a server (which will delay the rendering of the page). Other examples of uncontrollable parameters include the accuracy of an auto-complete algorithm, and the ability of an individual user to recall a command. *Critical parameter* is a subtype of uncontrollable parameters. A critical parameter is an uncontrollable parameter that is considered critical to the success of a system [571]. For example, page loading time can be considered critical for some web applications. *Sensitivity analysis* is used to understand how robust the user interface is in the presence of perturbation to these design parameters.

Collectively, all controllable and uncontrollable parameters relevant for a particular user interface span a multidimensional design space. An individual user that interacts with a specific user interface design forms an *operating point* in such a design space. The ideal operating point is the one where all design parameters align to provide the optimal user interface for the user given some criteria for what such an optimal user interface would provide to the user. However, it is nearly always infeasible to set the operating point at an optimal value. First, it is not viable for designers to be aware of all relevant design parameters. Second, many parameters are uncontrollable and therefore cannot be directly governed by the designer. Third, parameters are often conflicting for certain criteria. For example, enlarging the size of a button may make it easier for the user to click it at the expense of the button consuming more screen real-estate. This suggests that there are *trade-offs* in any user interface design, and an informed user interface design process has considered these trade-offs before choosing a final operating point.

This latter point motivates this part of the book. There are essentially two ways that an operating point can be set. First, it can be set *explicitly* by the designer. This means that the designer is aware of *most* relevant controllable and uncontrollable parameters of the design and trade-offs are therefore carefully considered in the design process. Second, the operating point can be set *implicitly* by the design itself. This happens when the designer is unaware of critical controllable and uncontrollable parameters of the user interface. This may still result in an acceptable user interface but it introduces a higher risk that the user interface may not perform as well as initially hoped and, in some cases, can lead to dangerous user interfaces that can result in financial losses, injuries, or even death. In reality, no designer can be aware of *all* parameters of a user interface design. It is therefore not only unrealistic to expect optimal user interface designs—it is unrealistic to even expect fully informed user interface designs.

A central role of this part in the book is to examine the various components of a user interface and discuss controllable and uncontrollable components that may affect users' ability to carry out their goals effectively, efficiently, and safely. By doing so, it is possible to develop an awareness of controllable parameters that can be set, optimized, or tuned, and uncontrollable parameters that are not directly governable but nevertheless may affect how users can interact with the user interface. Only after achieving this awareness is it possible to make informed choices about inherent trade-offs and ultimately be able to design a user interface that strikes an excellent balance in determining various parameters

Parameter	Type Explanation
Manufacturing Cost	Controllable
Sensing technology	Controllable
Layout	Controllable
Nominal key size	Controllable
Typing Speed	Uncontrollable
Typing Style	Uncontrollable

Table 23.3.: Examples of high-level design parameters for a physical keyboard.

governing its ability to support users' goals as effectively, efficiently, and safely as possible. Table 23.3 lists examples of such high-level design parameters for a physical keyboard.

23.4.3. QoC: Questions, Options, Criteria

Questions, Options, and Criteria, or QoC for short, extends design space analysis to cover reasons for design choices [494].

It is based on the concept of *design rationale*, a documented justification for the choice of design. An example: "Auditory feedback is provided for the completion of the task, because it is more likely noticed when driving and visual attention must be allocated to road." The idea is that an artifact is defined not only by its technical nature, but by the decisions that allow one to understand how it *could* have otherwise been.

Questions structure the space of alternatives, *Options* are possible alternative answers to Questions, and *Criteria* are used to evaluate and choose between options. For example, consider the scrollbar shown in **??** [1]. It has several features:

1. The scrollbar is normally invisible and appears only when a scrolling action is taken.
2. The scrollbar is narrow.
3. The scrollbar indicates the position of the view in the window.
4. The scrollbar indicates the relative size of the view in the window.

Each of these could have been formulated as a question. For example: *1. How to display?* Permanent or appearing. *If appearing, how should it appear?* When the cursor is moved or when a scrolling action is taken. *2. How wide?* Narrow or wide. *If wide, should it be of fixed width or relative to the size of the viewport as a proportion of page length?* and so on.

But how can one decide which option is the best? According to QoR, the options should be compared with the criteria, which should also be documented. For example,

1. Screen should be compact

[1]Based on original example of a scrollbar given by [494]

2. Scrollbar should be easy to find
3. Scrollbar should be easy to drag
4. Scrollbar should tell intuitively how much there is to scroll still.

The options should then be assessed against all relevant criteria. How do you do that? Evaluations can be based on experimental studies (Section 43) or analytical methods (Section **??**). Lacking such measures, they can also be based on previous experience, known good design examples, or reasoning. The study by Markussen et al. [504], for example, first argued about the design space of mid-air text entry and then empirically evaluated some candidate user interfaces that seemed promising, given the QOC analysis.

Summary

- User interfaces are what the user comes into contact with when using an interactive system.
- User interfaces comprise input devices, displays, interaction techniques, representations, and assemblies (also known as DIRA).
- There are five main styles of interaction: Command line, form, menu, and reality-based user interfaces.
- The design of user interfaces faces a tradeoff between different, conflicting criteria.

24. Input Devices

This chapter reviews input devices, the elements of user interfaces that sense users and use that information to operate computers. More precisely, an *input device* is a combination of hardware and software that senses users' movements, activities, and internal states to turn them into information in the interactive system. This information can be used to trigger a command or input a value, allowing users to interact.

The success of an input device relies on choosing the computation that should be carried out—the goal of the input device, and then ensuring that an appropriate representation of input is arrived at that can be sensed by the system. For example, a mouse is an input device that allows users to transmit their intentions about moving a mouse cursor on the screen. A mouse consists of a hardware device that allows the user to move it around on a desk and have its movements reflected in mouse cursor movements on the screen. A software driver ensures that the computer understands how to interpret the input signals that are transmitted from the hardware to the operating system.

Some input devices are familiar to us from everyday interaction, but others are experimental. **??** shows a longer example of the latter; consider these shorter examples, too.

- When you press a button on a keyboard, someone has designed the physical feel of the button. Oulasvirta et al. [612] studied different relations between the force users apply to a button and the displacement of the button, and showed that such relations greatly affect performance in something as simple as pressing a button.
- In Skinput, Harrison et al. [309] used an array of vibration sensors placed on the user's arm. These sensors allowed the sensing of contact on the skin, effectively allowing people to use their forearm as a touchpad.
- Microsoft Kinect was released in 2010 and used depth cameras to detect users' poses and gestures. By sensing poses and gestures, this input device allowed for a new class of games and interactions without controllers.

An example (Imaginary phone) is shown in the side box.

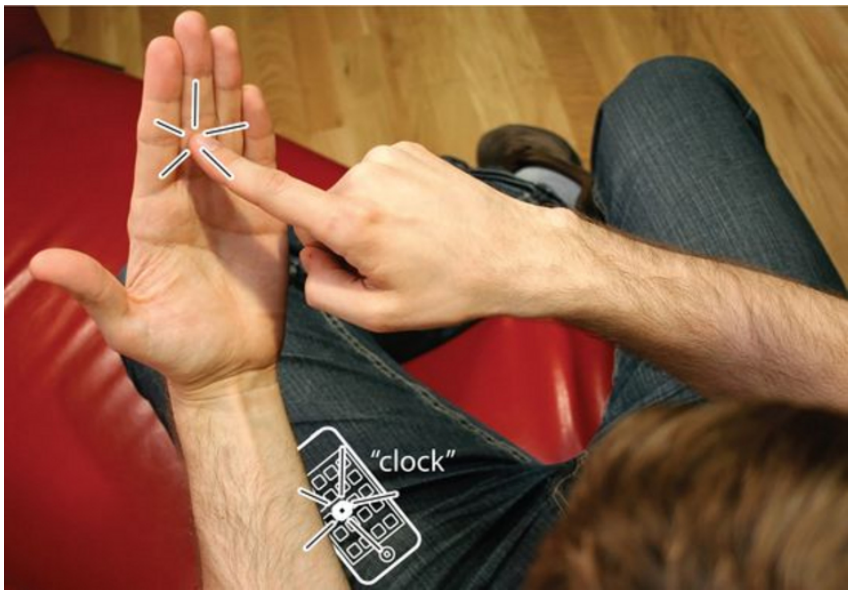
Recall from the introduction to this part that an input device differs from an interaction technique. Input devices sense users' movements and states; interaction techniques help users accomplish fundamental operations in the user interface using input devices and displays. Interaction techniques are discussed in Chapter 26.

Next, we discuss what exactly it means to *sense* something with an input device. Then we discuss input devices that pose few challenges to correct inference of the users' intentions, such as keyboards or pointing devices (e.g., a mouse). Thereafter, we discuss a class of input devices where it is much more difficult to figure out what the user is trying to input.

Paper Example 24.0.1 : Input in midair: The case of the imaginary phone

?? Most input devices we use rely on a surface on which something is moved. This is the case for the touchpad and the mouse. An active research area in HCI is input devices beyond rigid surfaces. Researchers have explored letting go of the requirement for an input surface, in so-called midair interfaces. The Imaginary Phone is a case in point. This research is based on the observation that our memory for the icon position on our phones works remarkably well (see Chapter 5). For instance, you can probably point to the message application on your phone even if all the icons looked the same. Gustafson et al. [300] departed from this observation to remove the need for phone's display.

In Imaginary Phone, the user's hands are tracked with a wearable camera so that input can be given by moving a finger and touching skin. No display is necessary after the basic layout is memorized. For example, an alarm can be set by drawing a gesture from a center toward the desired hour, as shown in the figure.



24.1. Principles of Sensing

Every input device relies on some type of sensing. A sensor is a device that transforms physical energy into an electric signal. Input sensing means something more than 'just' sensors, however. It refers to the whole sensing pipeline, including both software and hardware, that allows the computation of a command based on sensor data. An input device implements some sensing principle via hardware (e.g., sensor, microcontroller) and software (e.g., filtering, gesture recognition).

Input sensing has become a major topic of research in HCI. Thanks to improvements in micromachinery and microcontrollers, we can presently augment almost any physical object with sensing capabilities. For example, capacitive sensing techniques exist to enable the augmenting of plants, doorknobs, and even water with touch sensing (Paper Example 24.3.1 will give an example later in the chapter of augmenting plants with touch).

Yet, engineering a situationally appropriate input device is challenging. There is no universally best sensing approach. Consider an in-flight entertainment system, a crane operator's cockpit, and a GPS tracking device for boats. The movement capabilities and the user's goals, the constraints and capabilities of the system, and goals, as well as environmental factors, are vastly different. Users are good at noticing poor response characteristics, such as lag or inaccuracy. Careful engineering is needed to permit high user performance while being robust to environmental factors and cost-efficient. Consequently, symptomatic for this area is the long timeframes needed for R&D. For instance, the principles of computer vision-based hand tracking were investigated actively in the 1990s, but commercially successful solutions are coming out only now in VR systems.

Developing a good input device requires skills in software engineering, machine learning, systems engineering, and electrical engineering. In the following, we go through the steps of the input sensing pipeline.

24.1.1. Transducing

All sensing used in HCI transforms one type of physical energy into an electric signal. Traditional pointing devices sense motion, position, or force. An isometric joystick senses force. More generally, sensing principles used in HCI include mechanical (position, motion, rotation), electrical (capacitance, contact), sonic (ultrasonic, microphone, sonar), optical (proximity, light, image), radiation (radar, heat, temperature), magnetic (hall effect), and gravitational (e.g., LIGO). Also, chemical sensing methods are available, for example via moisture or via implanted sensors.

Analog sensors output an analog electrical signal, typically between 0 and 5 V. This is converted by an Analog-to-Digital converter (ADC), which determines the resolution of the output (e.g., 10 bits). While this interface is simple, it is prone to external noise. *Digital* sensors are more robust to noise. They output a digital signal: zeros and ones. To use this signal, a digital protocol is needed (e.g., UART, I2C, SPI).

For most physical events that we care about, there are several sensing principles. For example, touch can be sensed using a number of sensing principles, including capacitive, resistive, acoustic, infrared, vision, liquid crystals, planar scatter, and electromagnetic resonance. The popular capacitive touchscreens are based on measurement of capacitance or the measurement of energy stored in an electric field between two conductive objectives. A capacitor consists of dielectric material between two parallel metal plates, which responds to the changing distance of another dielectric object, the finger. The location can be sensed with many electrode pairs, the response characteristics of which are known. This sensing principle, unlike resistive sensing, its predecessor, allows multitouch sensing, but does not support pen input.

24.1.2. Sensing hardware

Each sensing principle must be implemented in a *device*, consisting of at least the sensor, a microcontroller, and a rig that keeps them together. These must be decided to strike a desired trade-off among objectives, for example, accuracy, battery life, latency, and robustness. In the case of capacitive sensing, one needs to decide the selection and configuration of electrodes to achieve the necessary accuracy and robustness to finger moisture and different pressing angles. The touch controller, a microcontroller-based chip, must be designed to translate the measurements to a computer-readable form.

24.1.3. Transfer functions

A sensor is a *transfer function.* A transfer function maps a change in input energy to a change in electric signal (output energy). We can learn a lot about a sensor by studying the properties of its transfer function.

The analysis of an input device starts by plotting the input energy against the output energy.

$$\text{output} \ = f(\text{input}) \tag{24.1}$$

For example, a regular push-button is based on a key switch that sends a break signal when the switch closes the circuit. From a transfer function point-of-view it transfers the contact force into the displacement of the key cap, the movement of which is resisted for example by a rubber dome. The key switch then transduces displacement into a step function, a simple on/off signal. We here introduce first basic terminology to analyze transfer function.

The transfer function offers a wealth of concepts to understand an input sensor. The *sensitivity* of a sensor refers to mapping of the change in the intensity of the input stimulation to the change in the output signal. A sensor has low sensitivity when a high range of input energy is mapped to only a few levels of output energy.

A sensor is said to be *linear* when this relationship is linear. *Linearity error* is the deviation of the output signal from a linear trendline $y = a + bx$, where a and b are empirically established. Some sensors have *non-linear transfer functions.* The relationship between input and output energy is nontrivial. Psychophysics methods can be used to find a suitable transfer function (see Chapter 3).

A related concept is *resolution.* Resolution means how many levels of output are expressed for different levels of input. This can be measured as entropy: $H = n \log_2 S$, where n is the number of levels of input and S is the size of the set of symbols in output. A low-resolution sensor loses information in the input signal. A high-resolution sensor is informally said to be *expressive.*

The *operating range* of a sensor refers to the minimum and maximum sensed amount of intensity. When input energy surpasses this threshold, the sensor is *saturated.* Under-saturated input energy is below the registration threshold of the sensor. Over-saturated input energy may destroy the sensor.

Bias level is a systematic source of error in transfer, and *noise* unsystematic (quasi-random) error. The difference is important to understand, because they are dealt with in

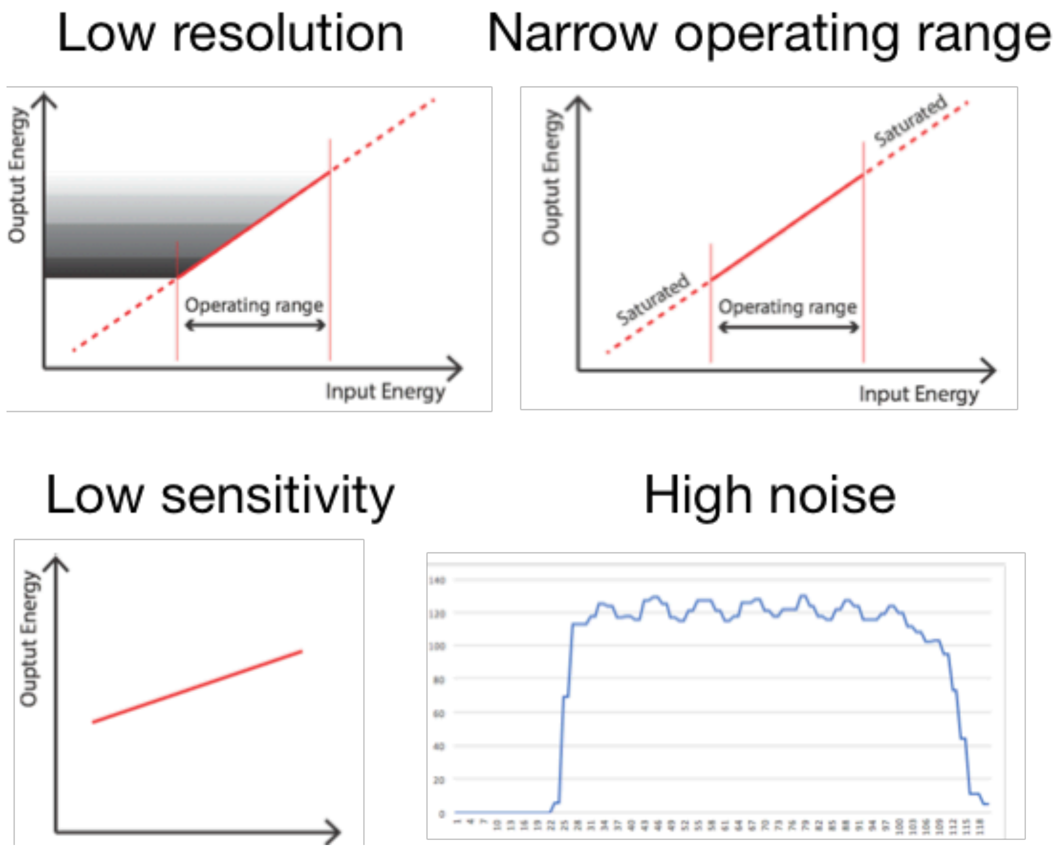


Figure 24.1.: Input horror gallery: Plotting the transfer function of an input sensor can expose a number of problems that hamper user performance. From left to right, top to bottom: Inability to distinguish among levels of control; Too narrow range; Inability to produce varied outputs; Noise. Images courtesy of Sunjun Kim

different ways. For example, computer vision-based sensing typically has high levels of noise, whereas mechanical sensing principles may have systematic bias.

24.1.4. Filtering

In filtering, the output signal of a sensor is suppressed in some way to get rid of noise or some other unwanted feature. Unwanted features in input sensing include noise (random variations in measured signal due to measurement noise), dropout (loss of measurement samples), drift (accumulating inaccuracy over time), and glitches (random spikes or other artifacts).

A key concept to understand how well filtering works is the *signal to noise ratio* (SNR).

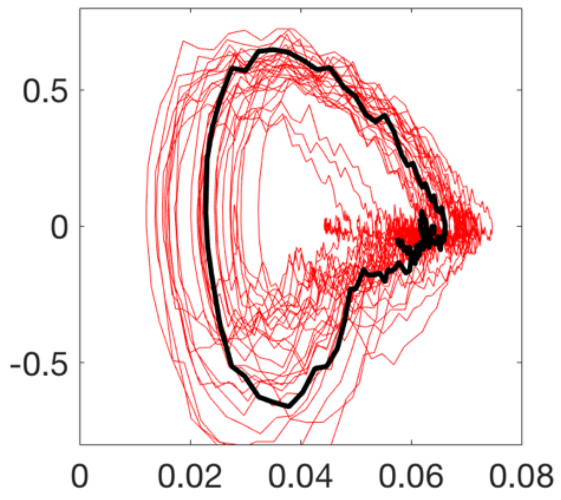


Figure 24.2.: Measurements of a button press in midair using a computer vision-based tracker shows significant noise which can be mitigated by techniques like filtering [613].

Intuitively, this is the proportion of noise in the power of a signal:

$$\text{SNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}} \tag{24.2}$$

The signal-to-noise ratio is often expressed in decibels: $\text{SNR}_{dB} = 10 \log_{10} \text{SNR}$. The SNR sets a limit on how much information can be 'saved' from the signal using filtering.

Frequency-based filters are the simplest filters. They suppress the signal based on frequency range. They can be used when noise occurs on a known range of signal frequency. Low pass filters pass only low-frequency signals, and high pass correspondingly the high-frequency signals. Band pass merges the two and passes only a specific frequency range.

Moving average offers another simple filtering approach. Assuming that noise has a mean of zero, a *moving average filter* can estimate the true value simply by averaging over a few previous measurement samples:

$$\bar{X} = \sum_{i=t-n}^{t} X_i \tag{24.3}$$

where $\bar{X}$ is the filtered value, X_i is value at i, t is current time, and n is window size. The drawback is that if the sampling frequency of the sensor is low, its response may feel 'laggy'.

Single exponential is a smoothing technique also known as first-order smoothing.

$$\bar{X}_i = \alpha X_i + (1 - \alpha)\bar{X}_{i-1} \tag{24.4}$$

where α is a smoothing factor $0 < \alpha < 1$. The idea of the approach is to control the contribution of older sensor readings. When the smoothing factor α increase, the filtered value follows closer to the latest sensor values. When it decreases, there is more lag but less jitter. In the *double exponential* technique, smoothing is performed twice to handle quadratic trends in the signal.

Speed-dependent filtering dynamically changes α based on an estimation of noise and velocity of movement. This is particularly relevant for continuous sensing, for example, a position sensor like a mouse.

Paper Example 24.1.1 : The 1€ filter

A key issue in input is dealing with noisy signals. The noise might cause an input value, such as the X-Y position from a touchpad, to fail to reflect the position of the user's finger. Alternatively, it may jitter, so that X-Y pairs that follow each other jump unpredictably.

While we may deal with such noise, we want to do so in a way that does not decrease the responsiveness of the device. Casiez et al. [139] found an elegant solution to this problem by what the called the 1€ filter.

The idea of the 1€filter is that the alpha gets smaller with faster cursor speed:

$$\alpha = \frac{1}{1 + \frac{\tau}{T_e}} \tag{24.5}$$

where $\tau = 1/2\pi f_c$ and $f_c = f_{c_{\text{min}}} + \beta|\bar{X}_i|$. The filtered value becomes

$$\bar{X}_i = \Big(X_i + \frac{\tau}{T_e}\bar{X}_{i-1}\Big)\alpha \tag{24.6}$$

The intuition of these equations is that cursor responsiveness remains high when the cursor is moved quickly, while accuracy remains high when the mouse is moved slowly.

24.1.5. Engineering challenges in sensing

Based on the above considerations, three recurring issues in engineering sensing may be summarized. The first concerns *what* can be sensed. The sensing modality of input refers to the particular input stream considered, such as speech, hand movement, pen movement, handwriting, gaze, etc. Input streams can also be multimodal which means multiple input streams are combined for the purpose of sensing user input. A useful input device must be robust to different interaction contexts and resilient to noise. Therefore, its sensing validity needs to be sufficiently high for it to be useful in practice where users will need to interact with the input device in a variety of setting under a variety of circumstances. The notion of sensing interaction potential captures the notation of expressiveness of the input device, the types of interactions an input device enables, and at what fidelity.

The second concerns *resolution.* The temporal resolution of an input device is its discrete resolution of the measurement of time. For example, a high temporal resolution of a mouse means that many samples of mouse movements are captured by the input device at a fixed unit of time. The spatial resolution of an input device is its discrete resolution of measurement in the spatial domain. For example, a hand-tracking input device may be able to track a user's hand with a resolution of 5 mm.

The third is *accuracy*. Accuracy reflects the ability of an input device to correctly sense the user's input. For example, a capacitive touchscreen will only be able to sense the location of the user's fingertip within some margin of error. Accuracy is modeled differently depending on the specific input device. For example, an input device can be thought of as a binary classifier only detecting *Input* or *No Input*, such as an eyebrow switch, is typically analyzed in terms of classifications that are either true positives or true negatives. Based on this it is possible to calculate accuracy but since accuracy is a composite measure it does not reveal as much useful information about the input device design.

In addition, as many input devices are realized in hardware, designing input devices frequently involves many considerations similar to other physical products, such as market demand, cost, development risk, certification, life cycle analysis, etc. However, while these considerations are certainly important for designing, manufacturing, distributing, and supporting input devices, they are not concerns central to HCI and will not be discussed here. Instead, there is a set of design factors that must be considered when creating a new input device in either software, hardware, or a combination thereof.

24.2. Keypads and Keyboards

Keypads and keyboards allow a user to input information, such as letters, symbols, numbers, and commands into a computer. There are many ways to design them, and their construction typically depends on their use case. As an example, an inexpensive method is to use a flat-panel membrane design.

Keypads

A *keypad* is a set of buttons typically arranged in a matrix layout. While the layout may vary, a common layout is the alphanumeric telephone keypad layout specified by the ISO/IEC 9995-8:1994 standard.

Keypads are common in household appliances, such as microwave ovens and calculators, dialpad phones, automated teller machines (ATMs) and machines in healthcare and manufacturing. They are appropriate for a large number of numeric entry and limited amount of alphanumeric entry.

Keypads gathered considerable attention during the early stage of mobile phone development, as the majority of mobile phones until the capacitive touchscreen era relied on used a design in which a physical ISO/IEC 9995-8:1994 keypad was coupled to a dot matrix display.

Keyboards

Similar to a keypad, a computer *keyboard* is a device that enables a user to input alphanumeric information into a computer system, such as text, numbers, and commands. Unlike a keypad, a keyboard typically contains all letter keys A–Z in the alphabet and in addition support a wide range of ancillary characters and functions. Most keyboards use a particular key arrangement known as the QWERTY layout (named after the first six letter keys at the top-right corner of the keyboard).

24.3. Pointing Devices

Keyboards allow users to enter text. Pointing devices allow users to select locations on a display for a variety of purposes, such as selecting or manipulating an object, or providing a drawing. Fundamentally, pointing devices can be subdivided into two classes: directly controlled devices and indirectly controlled devices.

A directly controlled pointing device is interacted with by pointing directly at the display. This allows the user to specify the precise location of the pointing in *absolute* coordinates. For example, using a directly controlled pointing device, the user can point at the top left corner of the display directly, which may in pixel space be, say, at location $(1, 1)$: the first leftmost pixel of the first top row of the display. This is an example of an absolute coordinate.

An indirectly controlled pointing device allows the user to specify a location on the display by manipulating the pointing device. In response, a cursor on the display moves as a function of the manipulation of the pointing device. For example, a user may use a mouse to control a cursor on a display. By moving the mouse, the cursor on the display changes its location. This is an example of *relative* movement. Control is indirect, as the user cannot directly select a desired location on a display by pointing at such a location. Instead, the user indirectly moves a cursor on a display by manipulating a pointing device. Relative movement can be modulated using a construct known as the control/display ratio, which is elaborated in the chapter on interaction techniques (Chapter 26).

24.3.1. Direct Control

Direct control pointing devices allow the user to select an absolute coordinate on a display.

Light Pen

One of the first direct control devices is the *light pen*. A light pen is a light-sensitive rod which the user holds in their dominant hand (see Figure 24.3). Light pens were designed to be used together with cathode-ray tube (CRT) displays (see Chapter 25). As explained in detail in the chapter on displays (**??**), a CRT generates a pixel image using a raster scan row by row, column by column. This means that the light-sensitive component of the light pen can be used in combination with the timing information of the raster scan to detect the location of the tip of the light pen.

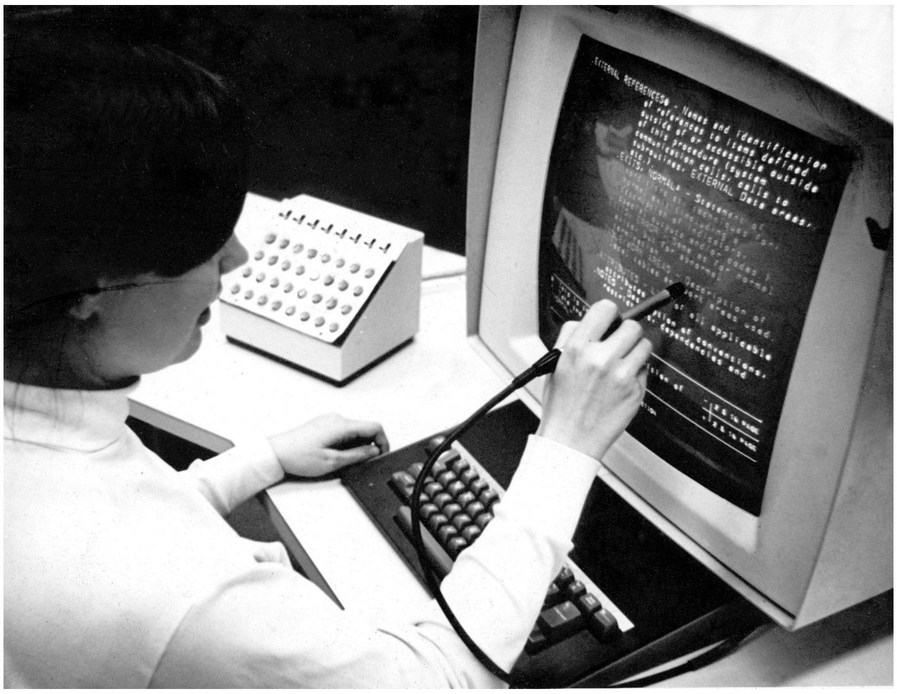


Figure 24.3.: A user interacting with an early hypertext system using a light pen. Image by Greg Lloyd under Creative Commons license.

As shown in Figure 24.3 a light pen allows the user to directly select a specific location on a display in absolute coordinates. In the figure, the light pen is used to select links in an early hypertext system.

Resistive Touchscreen

Another method to allow direct control is the use of a *resistive touchscreen.* Figure 24.4. At a high level, a resistive touchscreen is usually designed with two flexible sheets separated by an insulator, such as an air gap. When two sheets are pressed together using, for example, a tip of a pen or a finger, the two sheets make contact. By ensuring the sheets have horizontal and vertical traces it is possible to determine the touch location.

Resistive screens were popular in early mobile device design, such as personal digital assistants (PDAs), Tablet PCs, and early touchscreen mobile phones. Just as light pens they provide users with the ability to precisely select a location on a display by directly selecting the location of interest on the display. However, unlike light pens, resistive screens can be easily manufactured in small sizes suitable for portable electronics. Resistive screens are also inexpensive to manufacture and can be used with any implement that is capable of pressing the two sheets together, such as a glove, which typically cannot be readily sensed by a capacitive touchscreen. A pen-operated resistive touchscreen may also provide a higher sensing resolution.

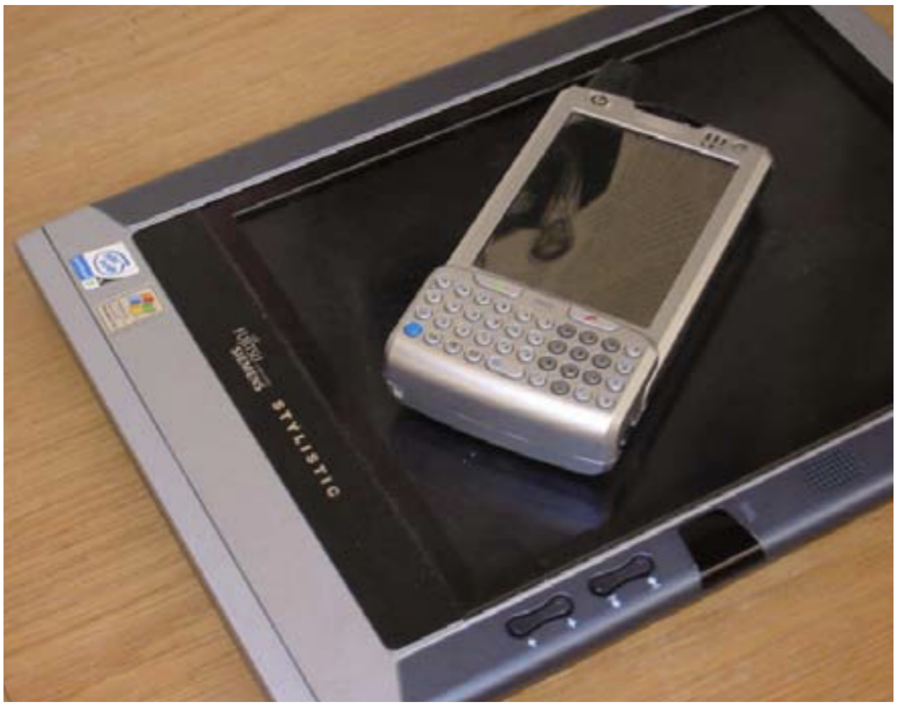


Figure 24.4.: A Tablet PC with a touch display and a personal digital assistant (PDA) with a resistive screen.

Capacitive Touchscreen

The *capacitive touchscreen* is ubiquitous on mobile devices. While capacity touchscreen technology has been available for many years, capacitive touchscreens were popularized by the introduction of the Apple iPhone in 2007.

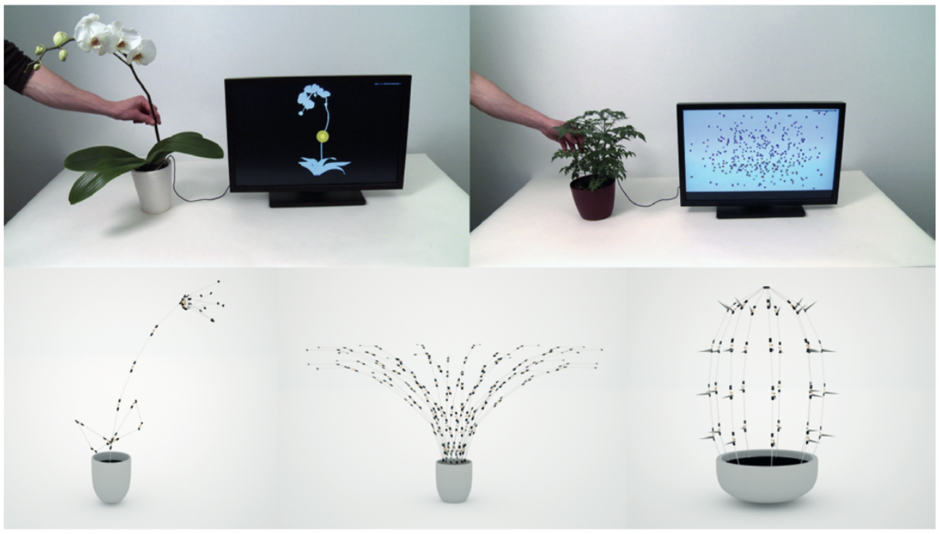
Capacitive touchscreens can be implemented in several ways. What they have in common is that they rely on capacitive coupling between an implement, typically the user's fingertip, and the display to sense the absolute position the user touches on the display.

Advances in capacitive touschreens have enables users to easily select precise locations on a touchscreen by lightly touching them with their fingertip. In addition, capacitive touchscreens allow multi-touch interaction, which enables a range of interaction techniques, such as pinching for zooming.

The typical tradeoff between a resistive and a capacitive touchscreen is that the response time is faster on a capacity touchscreen, but the precision is lower. In addition, a resistive touchscreen may not be as clear due to the dual membrane design.

Paper Example 24.3.1 : Botanicus Interactus

Interaction does not need to be limited to technological objects such as computers. Present-day sensing technology creates opportunities to augment everyday environments with capabilities for input and output. An example is *Botanicus Interactus* [656], which allows interaction with living and artificial plants.



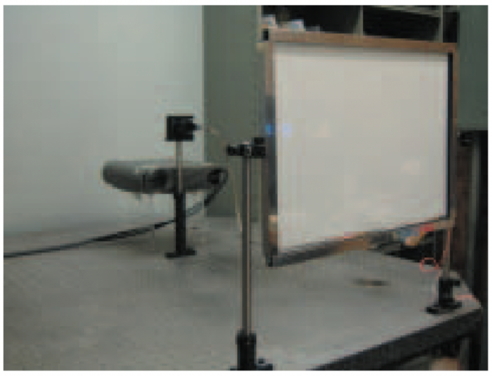
It works by exciting the plant with an electrical signal at multiple frequencies that span a range of 0.1 to 3 MHz. Regular capacitive sensing works by exciting a surface with a single frequency. Botanicus Interactus uses multiple frequencies to account for the more complex structure of a plant. Since the electrical signal path inside the plant varies by frequency, it is possible to infer touch locations by analyzing the frequencies which have affected the signal using machine learning methods.

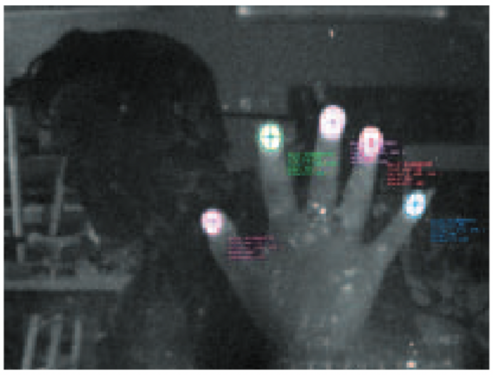
Paper Example 24.3.2 : Frustrated total internal reflection

In the early 2000s, it was expensive and difficult to design reliable large multi-touch surfaces. However, Han [305] demonstrated how it is possible to use a mechanism known as frustrated total internal reflection (FTIR) to create a low-cost, reliable multi-touch device.

Refraction is a term for the phenomenon in which light changes direction, or bends, when it passes from one medium to another. Total internal reflection happens when light passes through a medium with a lower index of refraction and the angle of incident of this refraction is greater than a critical angle. With the right choice of material, this interface at the medium can be designed to frustrate this total internal reflection, resulting in light escaping through this material instead.

A multi-touch FTIR surface exploits this operational principle. A typical design uses an acrylic pane which is edge-lit by infrared light-emitting diodes. Due to the total internal reflection, the infrared light is trapped within the sheet. A video camera is mounted behind the sheet, capturing the resulting image (**??** left). When an object, such a user's finger, makes optical contact with the sheet the light is frustrated, causing it to scatter through the pane towards the camera. This results in bright dots in the image which can then be processed using relatively simpler computer vision methods to determine the multi-touch locations right).





24.3.2. Indirect Control

Indirect control pointing devices enable the user to control a cursor on a display in response to manipulation of the pointing device. Thus, movement of the cursor on the display is relative to the manipulation of the pointing device.

Mouse

A *mouse* (**??**) allows a user to move a cursor by moving the mouse on a table. The physical movement of a mouse is translated into cursor movement. Several mechanisms are possible to enable a mouse. The first mice used two separate wheels below the mouse, which rolled when the mouse was moved on the table. A later design used a rolling ball

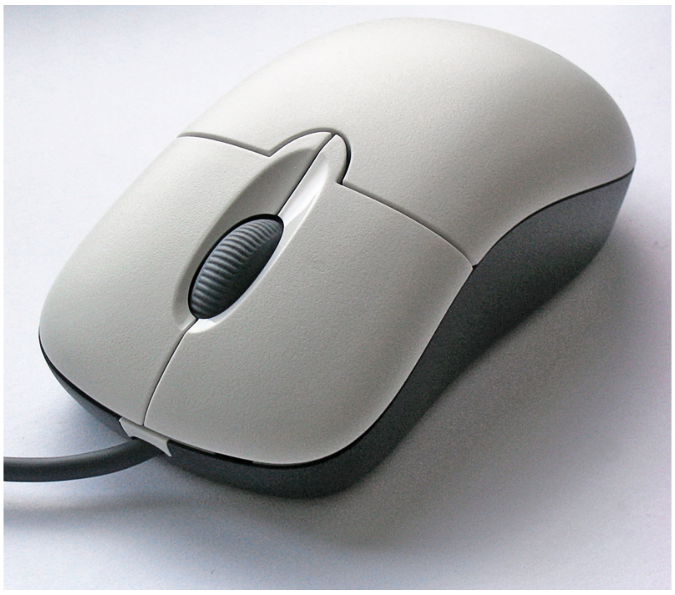


Figure 24.5.: An example of a computer mouse; image from https://commons.wikimedia.org/wiki/File:3-Tasten-Maus_Microsoft.jpg

design. Today, mice use optical sensors to determine relative movement without having to rely on moving parts in the electromechanical design.

A mouse typically also provides a set of mouse buttons that can be used to provide further information to the user. Another possible addition is the inclusion of a scroll wheel, which is typically used to allow the user to scroll the display up or down.

In addition to moving a cursor, a mouse can be used to perform gestures. For example, a common gesture is to drag and drop an object. Such and other mouse gestures are discussed in the chapter on interaction techniques (Chapter 26).

Touchpad

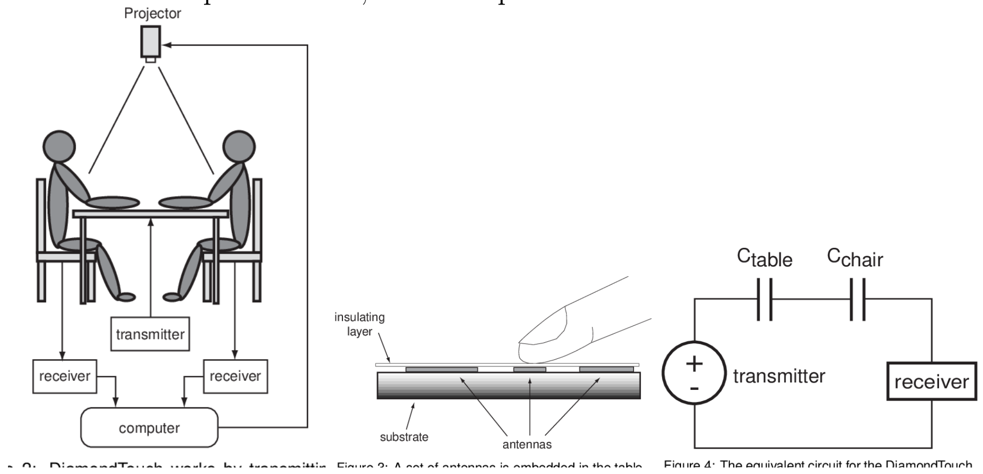
A *touchpad* is an indirect pointing device that allows a user to control a cursor on a display by moving a fingertip on a surface. The surface can be either resistive or capacitive. Touchpads can be augmented with additional features, such as tactile sensation feedback and button pressing mechanisms. In addition to controlling a cursor and similar to a mouse, a touchpad can also be used for additional purposes, such as for articulating 2D gestures.

Touchpads are common on laptops as a replacement for a mouse. However, unlike a mouse, a touchpad can be used in both relative and absolute coordinates. In other words, it is possible to map the input dimensions of a touchpad so that they are proportional to the display dimensions. This allows a user to, for example, select the topleft corner of the display by touching the topleft corner of the touchpad.

Paper Example 24.3.3 : Touch for several people at the same time?

As touch input devices become available, researchers began to wonder how to support touch for several people at the same time. That would require several touchpoints to be detected at the same time. It would also require that each touchpoint to be attributed to the correct person. Perhaps people would put objects on the surface (e.g., a pen, a pad) which should not be detected as a touch. How to do that?

Dietz and Leigh [191] developed the DiamondTouch table. As can be seen below (left), the table uses a projector that places an image on a table between the users. The table has been prepared with a number of antennas below an insulating layer (middle). The antennas works as switches so that it may be detected when a user touches within the area of the antenna (but not where). In the prototype of the DiamondTouch, there were four antennas per square centimeter. The identification of users were accomplished through instrumenting users' chairs (right). When the user touches the table, a circuit is completed from the transmitter, through the antenna on the table, though the user and the chair, and back to the transmitter. Because this circuit is unique to a user, the touchpoint allows the identification of the user.



DiamondTouch was later turned into a commercial product and was an early vision among many subsequent tabletop interfaces.

24.4. Uncertain Control

The previously discussed direct and indirect control input devices can deduce users' intention with a high degree of certainty. When any input is subject to noise, we use the term *uncertain control* to refer to a class of input devices that must be designed to be robust in the presence of uncertainty concerning the user's intention.

24.4.1. Switches

As a simple introductory example of uncertain control, we will consider a *switch*. An example of such a switch is a mechanical push-button, perhaps the most ubiquitous input device that allows binary control (on/off).

Push-buttons are electromechanical devices. When the user pushes the switch the user is forcing two electrical contacts together. The result is a low impedance (electrical resistance) path for current to flow. This sudden flow of current is used by the device to detect that the push-button has been depressed by the user. When the user let the push-button go the contacts are separated and current can no longer flow.

This operation is subject to noise due to a phenomenon known as switch bouncing, which means there is uncertainty around whether trailing mechanical motion has resulted in the switch to falsely trigger a second or third time upon activation. This is because the electrical contacts are subject to mechanical motion and will touch several times before assuming a rest position in which the contact is stable. The solution is to implement switch debouncing which eliminates this noise. Hardware buttons have such swich debouncing implemented in either hardware or software.

Another example of noise in switches is switch noise due to false activations and false detections. Some nonspeaking users with motor disabilities rely on a switch to communicate. One example is an eyebrow switch which activates when the user actuates their eyebrow. Such switches are prone to false activations due to noise, and hence require signal processing to reach acceptable levels of performance. In practice, such switches are still error prone and user interfaces reliant on such switches, such as specialized scanning keyboards for users with motor disabilities, have to be designed with such deficiencies in mind.

24.4.2. Accelerometers and Gyroscopes

An *accelerometer* is a device that measures the rate of change in velocity, that is, the acceleration of the device. It can be used for direct control or indirectly to sense what the user is doing. At rest, the measured acceleration will carry an offset corresponding to Earth's gravity. Accelerometers are commonly integrated into mobile phones and input devices because they allow for estimation of the orientation of the device. This information is frequently used to change the screen orientation between portrait and landscape mode, depending on how the user is holding the display. It also allows input. For example, gamers can control a character by tilting a mobile device. Another use of accelorometers is to use machine learning methods to infer user activity, such as the steps the user is walking over a period of time.

A *gyroscope* is a device that measures orientation and angular velocity. Input devices that use the functions provided by a gyroscope are typically implemented as micro-electro-mechanical systems. Since these devices measure device orientation and rotation they are frequently used to estimate device movement in 3D space and thus enable gesture control. It is also possible to combine data from accelerometers and gyroscopes to provide more robust gesture recognition and inference of user motion. Another potential example use of accelerometer and gyroscopic data is as a method for estimating the position of a Virtual Reality (VR) headset.

24.4.3. Brain-Computer Interface

It is also possible to allow the user to communicate with a computer without actuating any muscles. This is achievable using a *brain-computer interface* (BCI). A brain-computer interface translates the electrical activity in the user's brain into signals that a computer system can understand. Among its wider uses is as a prosthetic to control a robotic limb.

Brain-computer interfaces can be realized using a number of methods. A common non-invasive method is to electroencephalography (EEG) which uses electrodoes positioned on the user's scalp. The advantage of EEG and other non-invasive methods is their practicality as the demands on the user are lessened significantly although there is still a considerable set-up cost. The disadvantage is that the electrodes are positioned relatively far away from the brain and thus the signal-to-noise ratio is low. As a consequence, significant effort is required to ensure robust and accurate operation using such an input device to control a user interface. In practice, non-invasive BCI is severely limited in its use as an input device in HCI.

An alternative that achieves a much higher signal-to-noise ratio is brain implants. However, these techniques have serious implications for the user and have been used extensively to partially restore vision and movement in patients with disabilities.

24.4.4. Electromyography

An input device that is more reliable than BCI is *electromyography* (EMG), which senses the electrical activity in a user's muscles and translates this activity into signals that a computer system can act on. EMG devices vary in their construction, but their use in HCI is based on surface EMG which does not require the use of needles in the user.

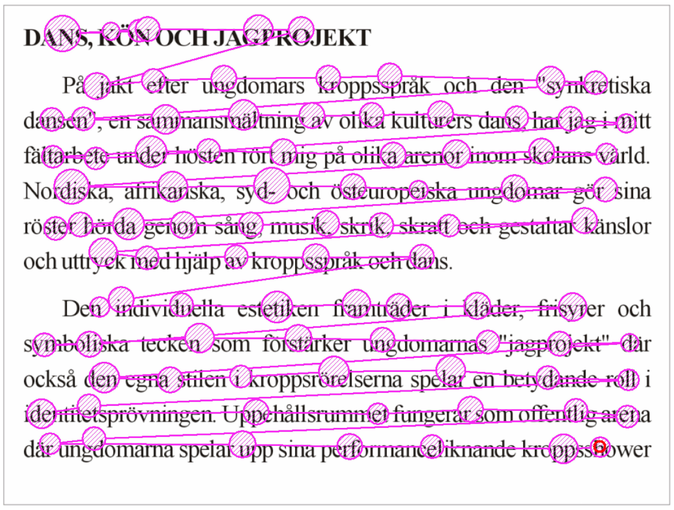
Surface EMG input devices produce signals that can be analyzed using signal processing and machine learning algorithms to infer user's activity. After processing, such information can be used to perform gesture recognition and, to some extent, be used to estimate fatigue.

24.4.5. Eye Tracking

An input device based on *eye tracking* estimates the user's gaze on a display. It does that by tracking the user's rotation of the eye and mapping that to cursor control. Several methods are available, and they all require calibration.

One method is to project infrared light towards the user's eye. The anatomical parts of the eye that reflect the light efficiently generate so-called Purkinje images. These images can then be used to infer the user's gaze. High-accuracy commercial eye-tracking systems typically implement some variant of this or a related scheme.

Another method is to optically detect users' gaze via a regular camera. The gaze position is inferred using computer vision methods. These methods are very sensitive to users' head movements and as a consequence are not very practical. While all eye tracking methods require calibration, optical methods may require recalibration whenever the user is significantly moving their head.



DANS, KÖN OCH JAGPROJEKT

På jakt efter ungdomars kroppsspråk och den "synkretiska dansen", en sammansmältning av olika kulturers dans, har jag i mitt fältarbete under hösten rört mig på olika arenor inom skolans värld. Nordiska, afrikanska, syd- och östeuropeiska ungdomar gör sina röster hörda genom sång, musik, skrik, skratt och gestaltar känslor och uttryck med hjälp av kroppsspråk och dans.

Den individuella estetiken framträder i kläder, frisyrer och symboliska tecken som förstärker ungdomarnas "jagprojekt" där också den egna stilen i kroppsrörelserna spelar en betydande roll i identitetsprövningen. Uppehållsrummet fungerar som offentlig arena där ungdomarna spelar upp sina performanceliknande kroppsshower

Figure 24.6.: An example of measured saccades and fixations for a reading task. Image source: `https://upload.wikimedia.org/wikipedia/commons/e/ef/Reading_Fixations_Saccades.jpg`

Electrooculography (EOG) can also be used. Electrodes are placed above or below or to the left or to the right of the eye. The electrodes measure the resting potential of the retina. If the eye is moved away from the centre position towards an electrode, then the electrode will sense this change. This information can then be used to infer relative eye position changes. A problem with EOG is drift. Another downside is the required set-up before use.

Finally, it is possible to carry out eye tracking by fitting the user with contact lenses coupled with some form of sensor, such as a magnetic field sensor. As the eye rotates, the contact lens rotates with it. This method enables highly accurate eye movement measurements. However, this technique is also invasive and requires set-up.

Eye tracking data is processed by considering saccades and fixations (see Chapter 3). A *saccade* is a ballistic jump performed by the eye. These are preplanned and take 150-250 ms to plan and execute. A saccade is followed by a *fixation*. A fixation is the period of time after a saccade when the eye is relatively stationary. They typically last between 200 and 300 ms. An example of a series of saccades and fixations are shown in Figure 24.6 for a typical reading task.

A sequence of saccade-fixations forms a *scanpath*. Scanpaths can be recorded using eye trackers capable of estimating the user's gaze locations at a certain, typically 30–60 Hz. Once sensed, such scanpaths can be aggregated according to location and time thresholds into areas that attract user fixation. The assumption is that if a user has fixated on a particular area it will influence the user's further actions or the details that the user is registering in memory. This follows from the *eye-mind hypothesis*: what people are looking at is what they are thinking about [900]. In other words, visual attention can act as a proxy for mental attention. However, for this to be likely to be true the user needs

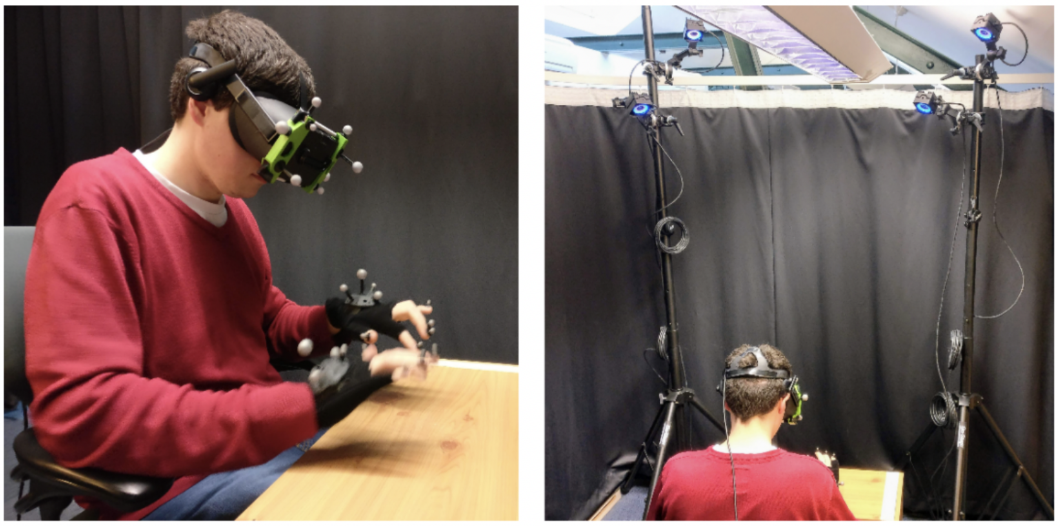


Figure 24.7.: An example of using optical markers to measure users' hand movements in typing tasks in mid-air [212].

to be given a suitable task.

Eye tracking is used in several applications. It can be used to carry out market research and usability testing where information on what the user is looking at is valuable for analysis. Eye tracking can also be used as part of an interaction technique, for example, to enable typing using so-called eye-typing methods, which are discussed in the chapter on interaction techniques (Chapter 26).

24.4.6. Hand- and finger-tracking

Hand- and finger-tracking is used to estimate the location of the user's hands and fingers in 3D space. Several technologies are available for hand- and finger-tracking.

Accurate hand- and finger-tracking can be based on optical markers. Such markers are fitted to the hands and fingers, or other body parts that need to be tracked. These markers are then tracked by high-resolution cameras. Software is then able to estimate the 3D locations of the markers in space. Optical marker-based systems are common in HCI research when there is a need to acquire a relatively high accuracy and low latency view of users' movement. For example, a study of ten-finger touch typing in mid-air may use an optical marker system to ensure the system captures as many nuances of typing behavior as possible (see **??**).

An alternative way of performing hand- and finger-tracking is to use a depth camera mounted on a headset, and then use computer vision methods to infer a hand skeleton. This approach is common in state-of-the-art virtual and augmented reality headsets. The skeleton serves as a low-dimensional representation of the user's finger, and hand movement can then be used more easily to, for example, design a gesture control system.

24.4.7. Microphone Input and Speech Recognition

Another input modality is the sound generated by users or their surroundings. One obvious example is voice commands, which allow a user to speak instructions directly to a computer. Sound is sensed using a transducer that converts vibrations of acoustic waves, sounds, into electrical signals. This transducer is called a microphone. Microphones can be connected into an array, called a microphone array, to allow sound to be more easily localized. Localization of sounds open up interaction possibilities, such as directing user feedback to a particular location.

Thus microphone input does not necessarily have to involve the user's voice. An early commercial example of another use is the Japanese version of the adventure game *The Legend of Zelda* in 1986 for the Japanese Nintendo Entertainment System (called Famicom). The Famicom arrived with an integrated controller that apart from a directional pad and a few buttons also had a built-in microphone. The Famicom was too underpowered to perform speech recognition but it could use this microphone in other ways. In The Legend of Zelda it was used to allow the player to defeat a particular enemy that was depicted with particularly large ears. By making a loud noise in the microphone these enemies would disappear. This is an early demonstration of the possible creative ways to utilize a microphone to provide new ways of input.

Microphones can be used to perform speech recognition. Speech recognition works by decoding observations of user's utterances into hypotheses of what users are intending to speak. Speech recognition can be provided by isolated words or it can be continuous, allowing users to speak isolated words, phrases or sentences. Speech recognition is commonly used for voice control but can also be used as an alternative text entry method by allowing users to perform dictation.

24.5. Expanding the Limits of Sensing

In the early history of computing, input devices mostly buttons and applied switch-based sensing. A mechanical switch affords high certainty in input; only rarely would a user accidentally activate a switch. Later, input devices began to be used to sense movement beyond an on/off event; the mouse and the joystick are prime examples. Although their input-sensing pipelines are more complex than that of a switch, because they need to deal with issues like noise and uncertainty, they fundamentally are about the sensing of movement.

How far can we push this paradigm? Many HCI researchers study alternative methods for sensing movement. For instance, an important research direction has been to manufacture input device that can be worn on human body. iSkin [859] is a skin-compatible tattoo (see Figure 24.8). It is a thin overlay that is transparent, flexible, and made of biocompatible materials. The authors propose methods for producing iSkins in variable sizes and shapes so that they can be worn on different locations of the body such as the finger, forearm, or ear. The sensor combines capacitive and resistive touch sensing. This makes it possible to sense touch even when the overlay is stretched or bent on skin. Furthermore, multiple touch areas can be supported, as well as more complex widgets,

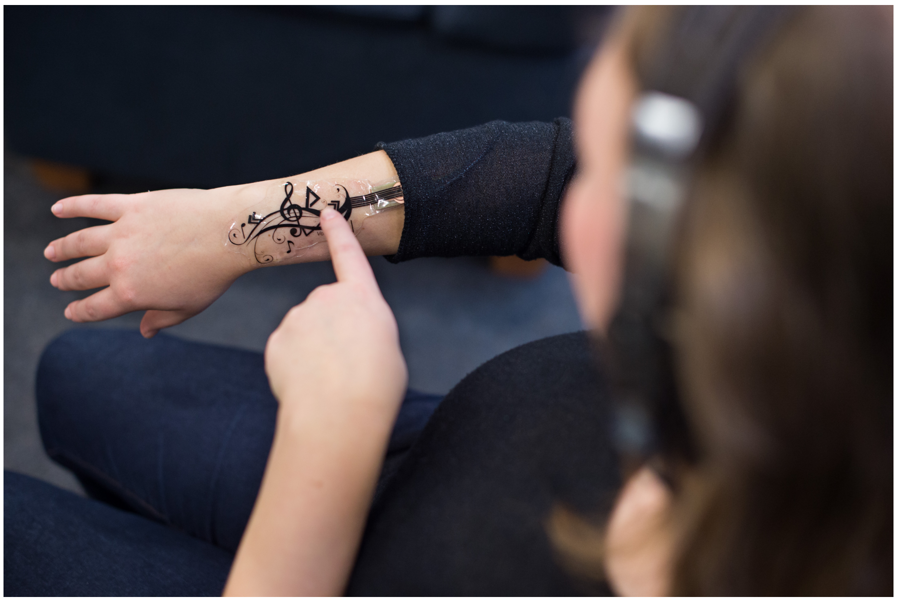


Figure 24.8.: A skin-compliant form of an input device [859].

such as sliders.

What if one wants to go beyond sensing movement? One strand of research on input devices aims to expand *what* we can sense. For example, there has been work on sensing temperature [664], mental workload [366], boredom [639], and much more.

Why would we want to sense such information? One reason is *adaptation*, the ability to change the user interface based on sensed information about the current state of the users, for instance, their workload. If the workload is high, we could simplify the user interface. Another reason is to create so-called *non-command* interfaces [573]. The idea of such interfaces is to infer the user's intentions through precise sensing, and thereby dispense with the need for a user explicitly issues a command. If you recall Norman's seven-stage model presented in Chapter 18, you will realize that this would dispense with the specifying and execution of actions in the user interface.

The fundamental problem in all these cases is whether the inferences to sensed phenomena or to intended commands are accurate. While this has sometimes been the case in controlled settings, making sensing robust in an uncontrolled setting remains hard.

Summary

- Input devices allow users to provide information into a computer system. At a high level, they involve three levels: computation, representation, and implementation.

- Design factors for input devices include their temporal and spatial resolutions, accuracy, sensing modality, sensing interaction potential, sensing validity, integrality and separability, and accessibility. The choice of appropriate input devices will typically require a design decision involving these factors.

- Input devices span a wide range but can be roughly divided into key-based devices, direct and indirect pointing devices and uncertain control devices. These input devices provide different opportunities and challenges for design, and thus some are more suitable for particular interaction context than others.

Exercises

1. Pointing devices. Remote pointing may be necessary when users are interacting with a large display. What types of input devices would be suitable? Consider their pros and cons.
2. Accurate input. Consider an input device for a medical device that must ensure users are making very accurate input. Input needs range from providing numbers and text to controlling a cursor on a screen. Which input devices would be suitable for which types of input? Is the choice of input device sufficient to ensure high reliability? If not, what other elements of the user interface would need to be considered?
3. Comparing input devices. Make a table with the following design factors provided in the introduction in this chapter as the columns: temporal resolution, spatial resolution, accuracy, sensing modality, sensing interaction potential, sensing validity, integrality and separability, and accessibility. Now create rows where each row represents an input device and score it between 0–10 on how well it meets the appropriate criteria defined by the design factors. How would the following input device compare: light pen, capacitative touchscreen, mouse, and touchpad? Are there any tradeoffs? Are the design factors sufficient to decide?

25. Displays

A critical component of user interfaces is the output devices or displays that allow the interactive system to transfer information to the user. We use the term *display* to cover visual displays (e.g., a computer screen), haptic devices (i.e., displays that you can feel), as well as other devices that can render information in a modality that humans can perceive. It is important to choose and design the display to match the capabilities of the user and the requirements of the task.

Displays can take many forms; consider these three examples.

1. Squeezeback [650] uses pneumatically driven compression around the arm to send notifications to people. This display can indicate different qualitative sensations that users can reliably distinguish.

2. Andrews et al. [24] explored how large displays could help intelligence analysts think about information (see Figure 25.1). They showed that the larger real-screen estate positively influence how analysts thought about and made sense of data.

3. SensaBubble [732] can shoot scented bubbles of particular sizes at the user. Information may be projected onto the bubbles until they burst. This allows for multi-modal information display in a playful form.

4. In 1969, Bach-y Rita et al. [33] published a paper on the tongue-display unit (see Figure 25.1. Because the tongue is packed with receptors, it is possible to represent visual information on it by using an array of 12 times 12 electrodes placed on the tongue.

In sum, displays are not limited to common displays that use LEDs (light-emitting diodes).

The defining objective of a display is to transform digital information into analog information so that people can accurately perceive it in one or more sensory modalities. To understand displays, we need to consider again the human perceptual system (see Chapter 3). A display needs to communicate digital information that matches what humans can perceive: There is little point in generating sound that we cannot hear or visuals with a resolution so high that our visual system cannot perceive it. Recall that our perceptual system has a range of limits. A well-functioning display ensures that the intended information becomes available to the user, rather than a biased or incomplete version thereof.

Let us give an initial example of the complexity of displays by considering color on visual displays such as a computer monitor. Let us assume that the job is to show how full the battery is. How would you choose the colors to show the battery level? The answer is surprisingly complex. Using a rainbow color scale is a poor choice. There is no

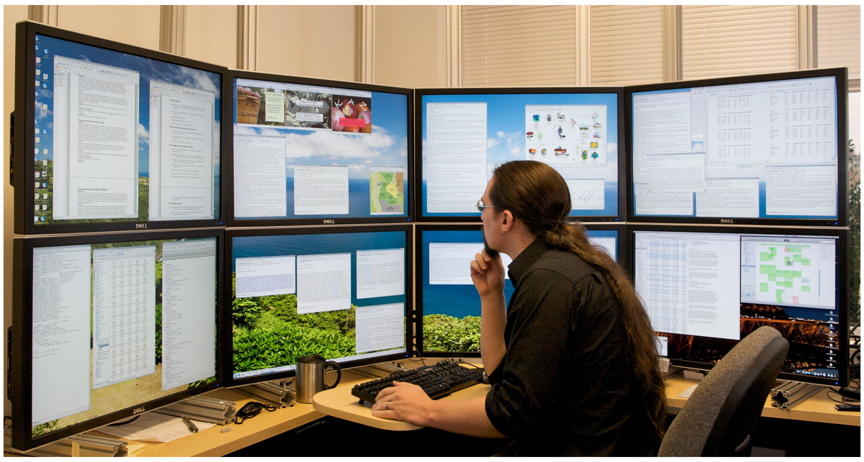
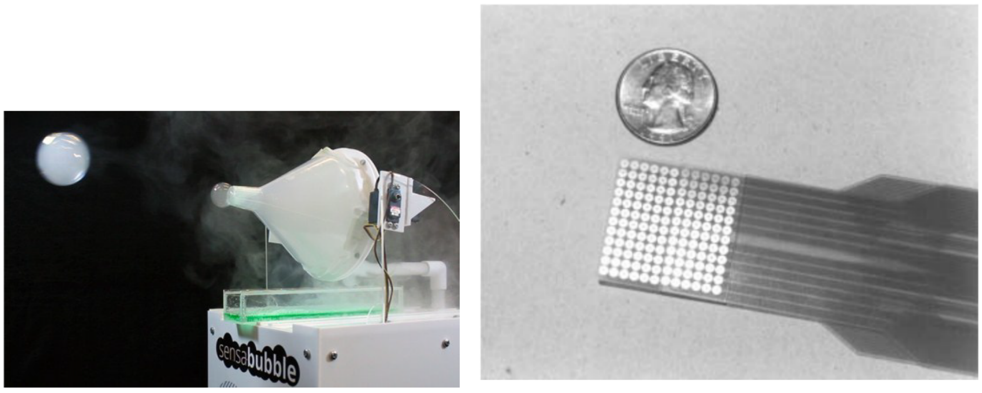



Figure 25.1.: Displays are not limited to the familiar touchscreen. HCI studies different arrangements and technologies for presenting information to users. Top: a large display used for sensemaking of work and data [24]. Bottom left: a multimodal display [732]. Bottom right: an electrotactile display to be used on the tongue [33, 34].

natural ordering of colors from red to violet that people easily recognize. Furthermore, picking different hues (such as red, yellow, blue) does not work well because they do not indicate quantity (that is, it is not perceptually obvious if orange would be a 30% full battery and red 60%). The color value is a better choice (i.e., the intensity of the color). But then it turns out that you cannot just halve the value of color encoding (say, its red-green-blue value) because humans would not perceive that value as half of the original value (it is not perceptually linear). Luckily, there are tools to help with this (such as ColorBrewer). The lesson is that designing displays is complex.

This chapter reviews common displays. We first discuss the central issues in encoding and rendering information: These are essential processes in turning digital information into something people can perceive. Then we discuss different devices working across different modalities, explaining their working principles and uses in HCI, and have to make informed design decisions when considering a set of output devices for a particular

HCI design problem.

25.1. Encoding and Rendering

Encoding concerns the basic operation of turning digital information, be it a name or a value, into something that a display may show. Many decisions in encoding are trivial: showing the digital content "v" on a screen is straightforward. But many decisions are not, in particular, when we go between modalities or when we encode something into a form that it does not naturally map to. For example, the auditory representation of a visual symbol, such as an icon, is not obvious. Brewster and Brown [95] created tactons, haptic versions of messages typically shown as dialog boxes or icons. This included carefully calibrating the frequency, amplitude, and duration used to encode the icon.

Rendering is the operation of transforming the encoded information into something that can be perceived by people. The issue here is that the rendering by display hardware may influence what people perceive. Early displays for virtual reality, for instance, had sufficiently low refresh rate that some users got motion such from the lag between their head movement and the visual scene. The rendering part of displaying information can therefore influence the experiences that users have.

Next, let us look more closely at encoding and rendering. We start by considering what information can be rendered.

25.1.1. Types of Value

A frequent point of departure for encoding is to separate different types of value. The types of values to be displayed determine the appropriate encoding. Three types of variables can be distinguished.

- Nominal variables (sometimes called categorical variables). Such variables do not have an inherent ordering or mapping to numerical values. They include the names of people, zip codes, and eye color.
- Ordered variables. Such variables may be ordered. They include the level of education and the satisfaction rating of user interfaces.
- Quantitative. Such variables are numeric and may be subject to arithmetic. They include, for instance, temperature, pulse, and level of rainfall.

These are not the only types of variables that we might want to output, but the three alone cover numerous cases.

We spend time considering what kind of value we want to display because it matters. As hinted at the beginning of the chapter, quantitative information should not be shown using changes in hue (e.g., 1, 2, 3). However, it is perfectly fine to display nominal values this way (e.g., Amsterdam, Brussels, Copenhagen)). In the first case, the colors should not be used on a map because it does not communicate the ordering and quantitative difference between the numbers. In the second case, color coding could be used, assuming that the mapping is known or available to the person viewing the display.

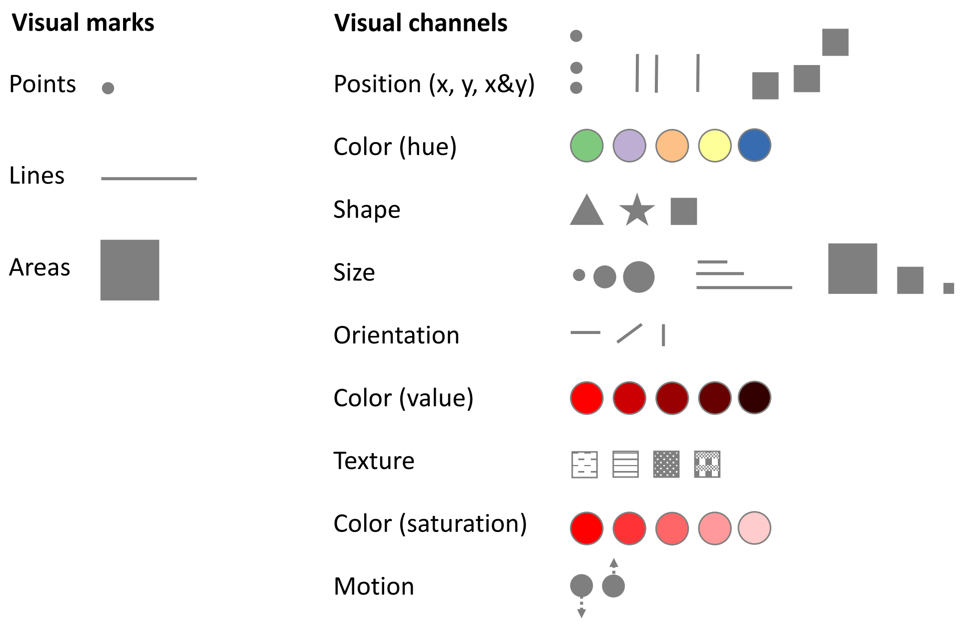


Figure 25.2.: This figure shows some ways of encoding data visually, based on Munzner [552]. Left is shown some examples of visual marks; right is shown some examples of visual channels.

25.1.2. Visual Encoding

Once it is known which values to show, the appropriate way to display these values can be chosen. Let us start by considering this for the visual modality, the most researched area of displays.

Visual encodings are typically separated into visual marks and visual channels [e.g., 552]. Visual marks are geometric primitives such as points, areas, and lines. Visual channels include the position of a mark, its color, and its texture. Figure 25.2 shows examples of marks and channels.

Before we discuss marks and channels in more depth, let us be more precise about color. The everyday language we use about color can be confusing. In technical terms, a color is considered to have three dimensions (see Figure 25.2 for examples).

Hue is how we usually speak about color. We refer to hue when we say "yellow" or "green". The RGB system for specifying colors relies on the hues red, green, and blue.

Value concerns the lightness or darkness of a color. It is sometimes called the luminance or brightness of the color. Value is thus the intensity of the color going from an absence of light (dark) to the full brightness of a color.

Saturation concerns the depth or intensity of the color. If we in a color picker sets one of the values of R, G, or B to full and the rest to none, we will have a fully saturated color. As we go towards a more even mix of R, G, and B values, the color becomes less saturated and will—when the R, G, and B values are the same—be white. Some books refer to this as intensity or chroma.

These dimensions of color work differently for encoding data, as we shall see next.

Effective Channels and Marks for Certain Variables

The area of *information visualization* has been concerned with how to show marks and which channels to use. It has established which types of marks and channels may be used for which variables and, crucially, which visual variables that do not work for certain variables. For example, the position channel is very well suited to show quantitative data. This is why scatter plots are suitable for quantitative data because they encode data as position on the axis. In contrast, the shape channel works poorly for encoding ordering because it has no natural ordering. Without other information, it is hard to decide if a circle is larger or smaller than a square. However, shape may be used effectively to encode nominal variables because our visual system effortlessly tells them apart.

How do we know that? Recall from the chapter on perception (Chapter 3) that psychophysics helps to study the perception of certain physical changes. Moreover, other types of experiments (Chapter 43) has also helped empirically characterize which shapes and marks work best. An early study is shown in the paper box below. Numerous subsequent studies—psychophysical and experimental—have taught us more about which encodings are appropriate [552, 322]. In particular, the following results are important:

- For *nominal variables*, hue is effective, as are shape and motion. Other channels, such as size or saturation, can communicate a difference in quantity that might be inappropriate. Thus, we could use hue to show different cities.

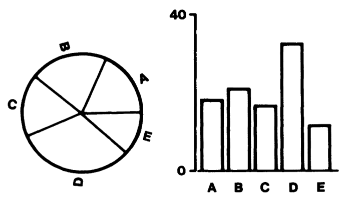
- For *ordered variables*, we can use value and saturation, as well as channels such as area and volume. In particular, the latter two are not effective for quantitative information, as it is hard to judge the exact difference in area (and hence the quantity). However, it is easy to see an order.

- For *quantitative variables*, we know that position, length, and orientation are highly effective. This is why scatter plots are easy to accurately read information from, and why comparing the magnitude of a variable in a bar plot is easy. Other variables, such as curvature, volume, saturation, and value may work too, but not as well as, for instance, position.

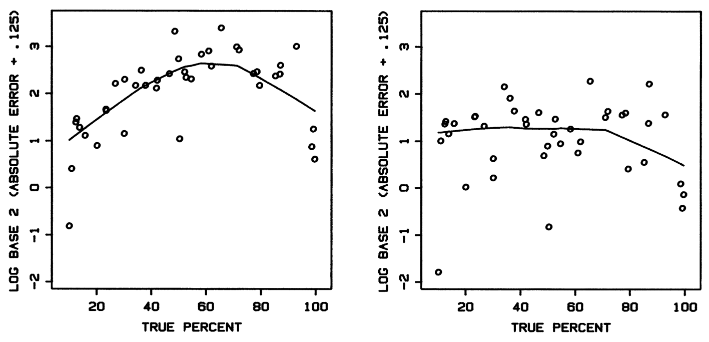
Paper Example 25.1.1 : Which visual channels works the best?

Although the use of graphs and other visual displays to convey information is many centuries old, the work by Cleveland and McGill [154] is considered seminal in understanding when we accurately extract information.

Their main innovation was to approach the use of graphs as a problem about how accurately people read information from the graphs (rather than, for instance, their aesthetic merits or their use in exploratory analysis of data). Cleveland and McGill [154] set out to understand which elementary tasks people perform when extracting information from graphs. These tasks are similar to what have been called channels. The figure to the right shows two graphs that rely on angle (or possibly area) and position along the y-axis to encode values.



The other part of the work of Cleveland and McGill [154] consisted of several studies that compare how accurately people perform these different elementary tasks. For instance, for the two tasks above, the data from an experiment with 54 participrequireiring them to state the percentage of a value relative to another, both represented in a particular way (i.e., as angle or as position). Across participants, the error rates can be plotted (y-axis below) relative to the actual difference (x-axis). The errors for the angle representation are shown to the left, whereas the representation for the position is to the right.



The conclusion here is that angle encoding results in more errors. In fact, only for very small or very large differences are the error rates for the encodings similar. Such data are the reason we know how to encode visual information so that people can read it accurately; they are also the reason that pie charts are considered problematic by many.

25.1.3. Encoding in Other Modalities

Above we have discussed in particular encoding for visual displays. The considerations for encoding in other modalities are similar. First, the encoding needs to match human perception so that the intended values are communicated and nothing else. Second, the encoding should be feasible with the display so that the intended values can be rendered.

As an example, consider encoding data using vibro-tactile actuators [825]. Vibro-tactile rendering can only be perceived by humans if the vibrations are between 20 and 500 Hz. However, people cannot separate more than nine frequency levels (and the recommendation is that the difference between levels be at least 20%. Some devices, like pneumatic-based actuators, can typically only render lover frequencies. This limits which type of haptic stimuli that can be rendered.

25.1.4. Rendering

In some cases, however, what we want to display already exists in an appropriate modality and there is no need to change it. For instance, we might have a 3D model that we wish to display in VR or a recording of a haptic experience to be replayed. In these cases, there is little concern about encoding, but only about how accurately a display can reproduce the model to a human perceiver.

As mentioned at the beginning of the chapter, both encoding and rendering are about matching the human perception. The *spatial resolution* of displays is often a factor that influence human perception. In visual displays, this is about the pixels per inch or similar measures; for haptic displays, it may be the number of actuators per inch (e.g., the number of pins in a pin-array display). The *temporal resolution* of displays are about their refresh rates, often measured in hertz. Finally, we may also consider *particular features* of the human perceptual system. In foveated rendering, for instance, higher resolution is offered at the spot where the user is looking because the periphery of the human eye sees less accurately than at the center.

25.2. Simple Displays

A light-emitting diode (LED) is a semiconductor that emits light when an electric current flows through it by releasing energy as photons. The amount of energy released determines the color of the light. LEDs are often used as indicators. For instance, in consumer electronics, a single LED is commonly used to signal that a device is turned off but connected to an electrical source, for example, the red light typically found somewhere on a television set. Computers and mobile phones with built-in cameras often include a LED as part of the camera circuitry that turns on when the camera is active, informing the user at all times whether the camera is active.

LEDs can also be arranged into LED displays where each LED represents a single picture element (pixel). Originally, such LED displays were limited in terms of color, as originally only red LEDs were widely available. Later developments resulted in green and blue LEDs. If a red, green, and blue LED are arranged in a triad, it allows a user to perceive it as a picture element (pixel). A LED display is a matrix of such red, green, and blue diode triads.

25.2.1. n-segment Display

An n-segment display uses segments that can be turned on and off to display information, typically numbers or letters. The number of segments can vary. One design is the co-called seven-segment display (see Figure 25.3), which uses seven segments to convey characters, typically numbers. A seven-segment display uses seven segments that can be lit up or not. This binary segment behavior results in $7^2 = 128$ possible states, although many of these states do not correspond to useful character representations. In general, an n-segment display can display n^2 possible states. Increasing the number of segments increases the number of potential states that can be expressed, and therefore increases the character set. In addition, an increase in the number of states tends to improve the ability to more accurately represent numbers, letters and other characters. In practice, n-segment displays tend to be designed with a slight shear to improve readability. In addition, the base segments indicated in Figure 25.3) are often complemented with additional segments representing the period and the comma at the bottom right.

n-segment displays are very common in consumer devices and appliances, such as microwave ovens, pocket calculators, alarm clocks and car stereos. The advantage of n-segment displays is low-cost of manufacture as the number of required electrical components is small. It is also easy to create an n-segment display as standardized decoder circuits for common segment display configurations, such as seven-segment and 14-segment displays, can be inexpensively procured and fitted into a device design. Well-designed n-segment displays may also represent letters, numbers, and other characters more clearly than a low-resolution dot matrix display.

The principle of having variable segments light up to indicate information has been known since the advent of electrical light. A segment display can be realized using any choice of switchable electric light, segment arrangement, and logic mechanism. A modern segment display is typically lit using light-emitting diodes (LEDs). The logic mechanism translates a predefined code for an individual character into the corresponding segments that should be lit to visualize the character. A seven-segment display with an additional segment for the period can represent all possible $2^8 = 256$ states in a byte (eight bits). A decoder circuit for a seven-segment display can then be implemented by providing eight input lines, one line for each bit, and eight output lines connected to the eight individual segments (ignoring any required electrical connections to provide power to the LEDs). In practice, four bits represent $2^4 = 16$ states, which is sufficient to represent numbers 0–9 in a typical decoder circuit, such as the Texas Instrument CD4511B decoder circuit.

25.2.2. Dot Matrix Display

An alternative approach to an n-segment display that provides higher resolution is a *dot matrix display*. An example of a rudimentary type of dot matrix display is a *flip-disc display*. Such a display consists of a large number of discs with a light color on the front and a dark color on the back. A flip-disc display then flips the discs to produce a dot pattern, such as a text message. They are commonly used in public spaces, such as large outdoor signs and publication transportation.

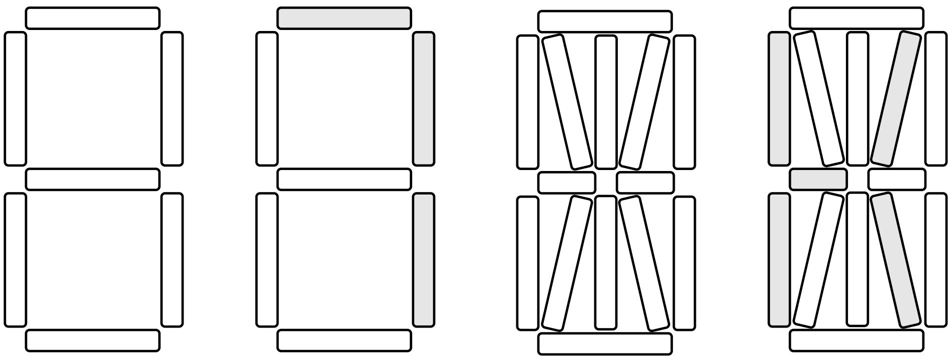


Figure 25.3.: Two examples of n-segment displays. The two displays to the left illustrate seven-segment displays with either all segments unlit (left) or visualizing the number '7' (right). Since each segment is lit or unlit, there is a total number of $2^7 = 128$ possible segment combinations that will be shown on the seven-segment display, although some of these segment combinations may not correspond to useful representations of characters. The two displays to the right illustrate 14-segment displays, with either all segments unlit (left) or visualizing the letter 'K' (right). A 14-segment display has a total number of $2^{14} = 16384$ possible states. If the segments are placed wisely, this allows better representation of letters and the ability to visualize many more characters.

25.3. Visual Displays

A *visual display*, such as a computer screen, allows a computer system to present information to the user. Visual displays have undergone tremendous development in the last decade.

25.3.1. Display Technology

Central to a display is the mechanism used to reveal information on a screen. The traditional approach, common until the beginning of the 2000s, is known as color CRTs (cathode ray tubes). It relies on three electron guns (one for each red, green, and blue color) that repeatedly scan a glass tube consisting of phosphorus dots in red, green, and blue. Each pixel position consists of three phosphor color dots, each emitting either red, green, or blue light when excited. When the phosphor dots are struck with the beam, they emit light. The electron guns scan the phosphor-coated screen in a *raster pattern*, drawing each line in the image from left-to-right and each line from top-to-bottom. The frequency of these screen updates is known as the *refresh rate*. A higher refresh rate tends to reduce perceived screen flickering.

In a CRT television set, the refresh rate is determined by the color encoding system chosen in the region. For example, PAL (phase alternating line) color encoding, common in Europe, South America, and parts of Asia, uses a 50-Hz-refresh rate and 576 visible lines. NTSC (national television system committee) color encoding, common in North America and Japan, uses a 60 Hz refresh rate and 480 visible lines. The difference in refresh rate is historical: when color encodings were introduced, there was a desire to align the refresh rate on CRTs with the AC (alternating current) power line frequency (which is either 50 or 60 Hz, depending on the region).

When the electron guns reach the end of a line, they are turned off and repositioned at the start of the next line. The delay involved in this operation is known as the *horizontal blanking interval* (often referred to as 'hblank'). Similarly, when the electron guns reach the bottom of the screen, they are turned off and repositioned at the beginning of the first line at the top of the screen. This delay is called the *vertical blanking interval* ('vblank'). These intervals became very important in early video game production, where typically the programmer could only update video memory during the vblank period due to cost limitations of chip sets at the time. Today, a technique known as 'double buffering' is used to avoid screen tearing. The central idea is that the system uses two buffers. The first buffer is used to directly perform graphical updates. When all graphics updates are complete, the first buffered is copied to the second buffer, which is displayed on the screen. This prevents the user from perceiving graphical artifacts, such as screen tearing and partial graphical updates.

Since the early 2000s, improvements in flat-panel displays have phased out the use of CRTs. Common flat-panel technologies include LCD (liquid crystal display), LCDs backlit with light-emitting diodes (commonly referred to as LEDs), plasma panels, organic light-emitting diode (OLED), and quantum dot light-emitting diode (QLED) displays.

25.3.2. Head-Mounted Displays

A *head-mounted display* (HMD) is a wearable display that can allows the user to be exposed to various degrees of virtual information. The *reality-virtuality continuum* [530] defines a spectrum of immersion for head-up displays (see Chapter 29). At one extreme end, the user is exposed to the real environment and there is no exposure to virtual information. At the other extreme end, the user is fully immersed in a virtual environment. This immersion can be achieved by wearing a virtual reality headset. It is also possible to augment the real environment with virtual information, for example, by allowing the user to wear an optical see-through heads-up display which can reveal virtual information that is mixed with the user's view of the real world. Depending on the level of immersion, the operating moves to the left and to the right in the reality-virtuality continuum.

Virtual Reality

For VR to be fully immersive, VR headsets (see Figure 25.4) must produce a very large virtual world that fully occupies the user's field-of-view. In addition, for information, in particular text, to be legible, the resolution needs to be high. The net effect is that the number of pixels that need to compute their pixel values is very high. This results in a

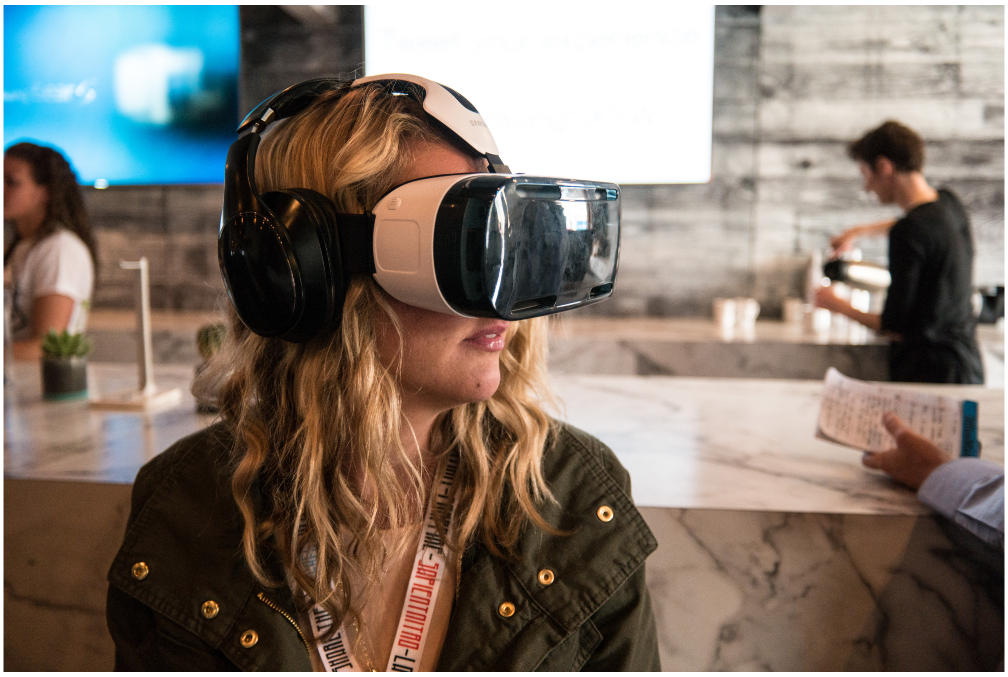


Figure 25.4.: A user wearing a VR headset. Photo by Nan Palmero, reused by Creative Commons License.

high technical demand on the ability of the VR system to render this 3D virtual world at a high resolution at a high refresh rate. In addition, when the user moves their head, the user expects to see a new part of the virtual world. Thus, the VR system must track the user's head movement and respond to this movement with low latency.

Since a virtual world is fully immersive but a VR system is unable to fully reproduce users' equivalent experience when looking around and interacting in the real environment, there is mismatch. This mismatch, exacerbated by the fatigue in wearing a headset over a longer period of time, results in simulator sickness. This effect is so prevalent that most empirical VR research with users routinely tests for simulator sickness. In addition to tracking the user's head movement, VR headset can allow users to walk around in a virtual world. This requires positional tracking of the VR headset both in terms of its Euclidean coordinate in the real environment and the direction of the headset—which way the user is looking. Several techniques have been developed to enable VR positional tracking including active markers (e.g., Valve's Lighthouse system).

Since VR enables users to fully immerse themselves in a virtual world, VR is eminently suitable for interactive simulations. Typical simulation domains where VR usage is widespread include training activities in medicine, such as virtual surgeries, defense, and manufacturing. In addition, VR can be used in the visualization of abstract information. Since VR is fully immersive, it allows complex visualizations to occupy a very vast virtual space. This may be beneficial in, for example, computational design, where users may want to be able to analyze complex information, such as the blade geometry of an airplane

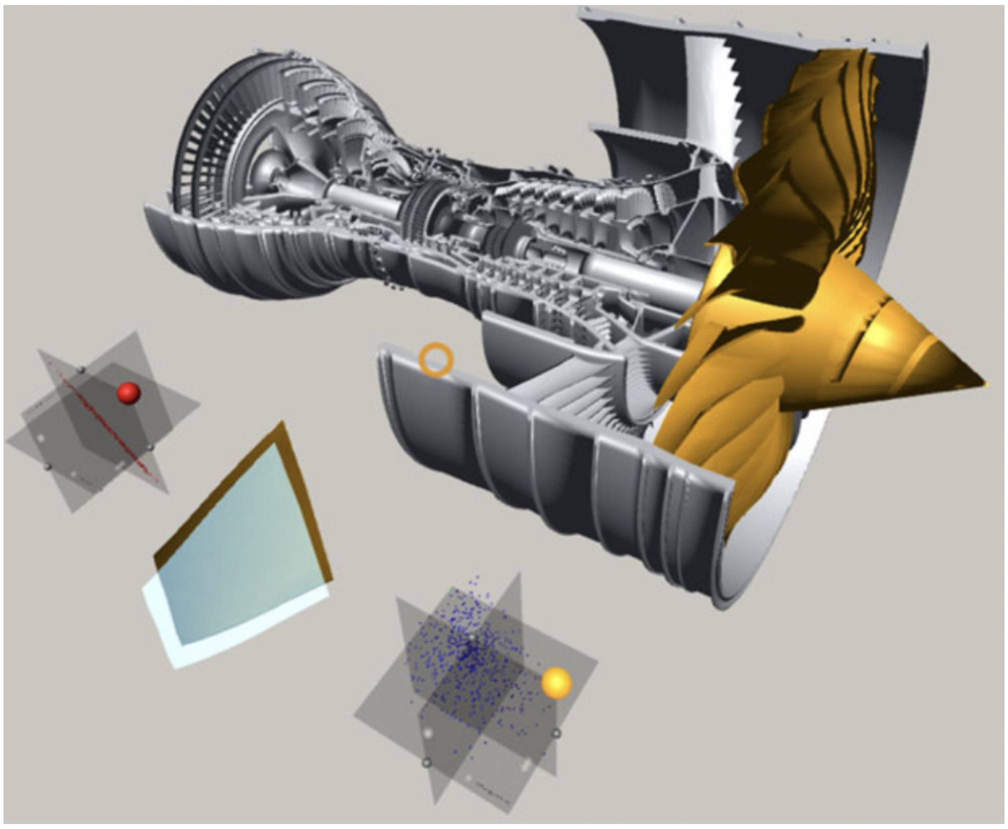


Figure 25.5.: A user inspecting the implications of a change in the blade geometry for an airplane engine in VR [794].

engine, and at the same time be able to see the consequences of parameter modifications (Figure 25.5).

Another example where VR may be useful is as a fully portable virtual office [293]. VR allows a user to wear a headset and consistently reconstruct an elaborate high-resolution real-world multi-display environment. In addition, since VR is in 3D the additional operational space allows common productivity applications, such as spreadsheets [271] and presentation software [73], to be augmented with additional visualization and interaction features.

Augmented Reality

Augmented reality can be provided through a headset which allows users to view the real world by merely looking around and moving their head and at the same time perceive virtual objects that appear to be located in the real world, a process known as registration.

AR headsets are commonly of two designs. The first is *video passthrough*, which means a camera is mounted at the front of the headset. The user views a video stream from this camera using displays mounted very close to the user's eyes. Virtual objects are rendered

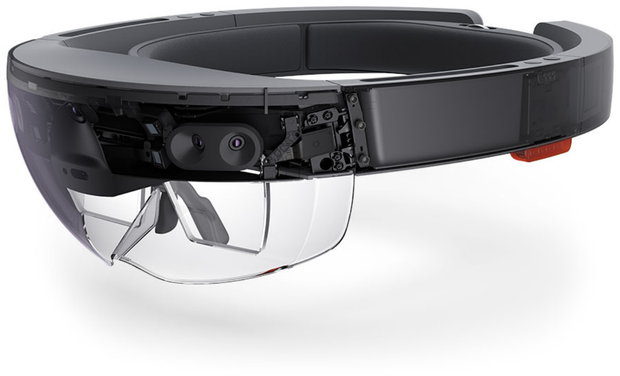


Figure 25.6.: An example of an optical see-through head-mounted display, the Microsoft HoloLens.

on top of this video stream. This solution is less expensive and technically difficult to realize, but results in considerably lower fidelity in the user's view of the real environment. Furthermore, it is challenging to correctly register virtual objects in the video stream.

An *optical see-through* head-mounted display (OST HMD) allows users to perceive virtual digital content seamlessly blended in the physical environment. The user perceives the real world through a glass pane, and virtual objects are registered to appear in the real world of the user using holographic technology. An example of such a headset is the Microsoft HoloLens (Figure 25.6). OST HMDs are advantageous in that the user perceives the real world through essentially a glass pane. Hence, little to no fidelity is lost. However, such displays have difficulties in generating accurate colors. For example, it is common to perceive rainbow bands on white panels. In addition, it is difficult to generate a wide field-of-view.

25.4. Audio

Humans perceive sound as vibration in air. These vibrations are called acoustic waves. An acoustic wave is received by the human auditory system and perceived as sound in the brain. A human can typically perceive sound in a frequency band between approximately 16 Hz and 16 kHz [548], although the specific range will vary between individuals and factors, such as age, can affect the perceived range.

There is a rich design space for *auditory displays*. In an analog electrical circuit, sound is represented as an electrical signal, a variation in voltage. The mathematical representation is a continuous time-varying signal; a simple example would be a sinusoidal signal. In a digital circuit, sound is represented as a series of binary switching states. The mathematical representation is a series of binary digits. An analog-to-digital (ADC) converter is a circuit that converts an analog signal into a digital representation. A digital-to-analog converter (DAC) is a circuit that converts a digital representation into an analog signal. Several electrical circuit architectures have been proposed in academia and industry to efficiently realize this functions. A speaker is a transducer that converts an electrical acoustic signal into sound pressure so that it can be perceived by humans.

Spatial sound

Humans can infer the direction and distance of an origin of a sound with some precision. Since humans typically have two ears, sound will arrive at the individual ears with some time difference. This is known as the interaural time difference, and the human auditory system is capable of exploiting this to enable sound localization. A monaural (mono) sound system either emits the sound signal to a single speaker or emits the same sound signal to speakers. This prevents any notion of sound localization. A stereophonic (stereo) sound system reproduces sound localization to various extent by emitting different sound signals to different speakers. A common example is a pair of headphones.

Spatial sound refers to the creation of spatial experience in listening to computer-generated sound. Consider for example playing a 3D video game, the sounds of which can be heard from the direction and distance they would occur in the virtual reality. *Sonification* uses spatial sound and other cues to cast visual or other information into sound. One could, for example, listen to an sonification of atmospheric impurities or display continuous variables in a dataset using sound.

Auditory icons and earcons

Auditory icons map an event to a sound-producing event on a computer. For example, the sound of a bell ringing may be a reminder of a calendar event. *Earcons* are auditory icons that do not resemble the event that originated the sound. They relax the requirement of mimicking the sound-originating event. This means that their meaning is learned by associative learning and takes time. Research has looked into the design of earcons by combining them and forming hierarchies of them by using shared cues like rhythm or pitch. Earcons are commonly used to represent temporal events, such as a notification for an incoming message or an error. Brewster et al. [96] suggested the following guidelines for designing earcons:

- ”Use musical instrument timbres, simple tones such as sinewaves or square waves are not effective. Where possible use timbres with multiple harmonics as this helps perception and can avoid masking”
- ”If listeners are to make absolute judgements of earcons then pitch/register should not be used. A combination of register and another parameter would give better rates of recall”
- ”Complex intra-earcon pitch structures are effective in differentiating earcons if used along with rhythm or another parameter”

- ”Make rhythms as different as possible. Putting different numbers of notes in each rhythm is very effective”
- ”Great care must be taken over the use of intensity because it is the main cause of annoyance due to sound”
- ”This may be stereo position or full three-dimensions if extra spatialisation hardware is available. This is very useful for differentiating parallel earcons playing simultaneously”
- ”When playing serial earcons one after another use a 0.1 second gap between them so that users can tell where one finishes and the other starts”

Recognizability poses limitations to earcons’ wider use.

Speech

Speech can be synthesized by computers, which has opened the door for conversational user interfaces. Speech can be used in many contexts, including digital assistants, announcement systems, and screen readers. However, until recently, limited accuracy of voice recognition has posed a problem. Of what use is a speech interface that one cannot talk back to? Especially punctuations (”Stop it!”) and proper names would cause recognition issues. A word that is recognized erroneously requires some other modality for fixing, for example, a touchscreen. Recently, advances in text-to-speech technology and language models are making it possible to use speech to command a digital agent.

25.5. Haptics

Haptic technologies enable users to perceive touch or, in general, forces on their body. A traditional application of haptics is telerobotics, where the user directly controls a remote robot. Haptics are then used to improve users’ ability to perceive reactions of the robot in response to control commands.

25.5.1. Vibration

A common approach to activate a sense of touch is the use of vibration. Vibration can be generated straight-forwardly using a *vibrator*. An electric motor transfers torque (rotational force) using a drive shaft. If an unbalanced mass is connected to the drive shaft, the system will produce vibrations. Such a system can, for example, be used to generate the rumble sensation commonly found in video game controllers and the vibration signal of mobile phones.

25.5.2. Force Feedback

A force feedback device is a system that generates a perceived force by actuation. An alternative term is force-reflecting interface [509]. One example of force feedback is a mechanical exoskeleton glove. When the user wears the glove the system tracks the user's fingers. If the fingers are supposed to interact with a virtual object then the system uses force feedback to simulate the forces the user would have experienced had the user touched a physical object. Mechanical exoskeleton gloves have applications in telerobotics, teleoperation and virtual reality.

Mechanical exoskeleton gloves can be grounded. Grounding means that the system is connected to a grounding point, such as a desk or a floor. This allows the system to simulate the sensation of weight, for example, a user lifting a heavy virtual bag in virtual reality. Another extension is to augment the gloves with additional feedback, such as vibration feedback or thermal feedback to simulate tactile sensation or touching cold or hot surfaces.

Massie et al. [509] set out three criteria for effective force feedback:

Free space must feel free The device should minimize encumberment of the user and minimize external forces when the user is moving in free space. This means the device should have low inertia and no unbalanced weight.

Solid virtual objects must feel stiff The maximum stiffness that a virtual surface can represent determines how solid an object will be perceived by a user. In reality, there is a limit to the stiffness a system is required to simulate in order for users to perceive a virtual surface as solid. For example, Massie et al. [509] states they have found users to perceive a stiffness of 20 N/cm as representing a solid immovable surface.

Virtual constraints must not be easily saturated Virtual walls and other solid immovable surfaces should be perceived as solid and immovable. The system should support the maximum exertion force of the finger suitable for the application, which can range from a force of 10 N for precise manipulation [509] to 40 N for the maximum exertion force of a human finger [791].

The braking mechanism in a force feedback glove can be constructed using a variety of techniques. One way is to use a pulley system where each finger is connected to a cable. A motor pulls the cable to simulate force feedback [509, 176]. The advantage of this design is its relatively simplistic actuation principle. A disadvantage is its complicated mechanical design, which makes a device expensive to build and maintain. Another disadvantage is the possible risk of the pulley system accidentally overextending a finger, causing injury.

Another example of a mechanism is to use a ratchet wheel construction, which allows the mechanical exoskeleton to freely follow finger movement. When force feedback is activated, a driving mechanism inserts a linear slider into the ratchet wheel and prevents the mechanical exoskeleton joint from moving (Figure 25.7). The result is a sensation of force feedback as the user's finger movement is stopped by the braking mechanism [299].

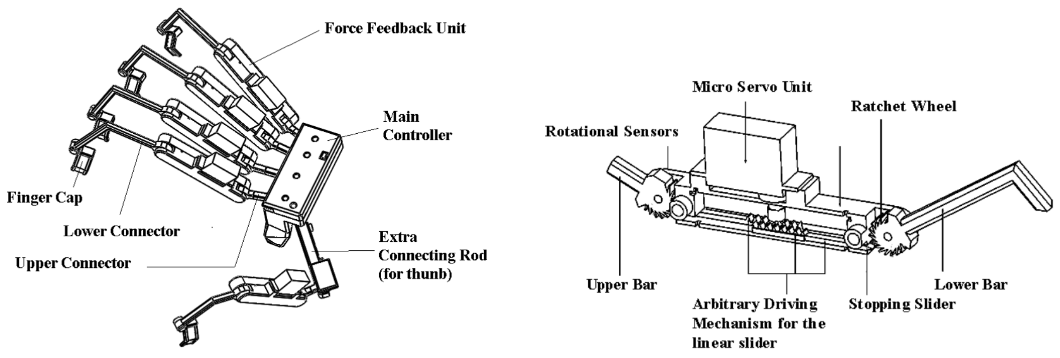


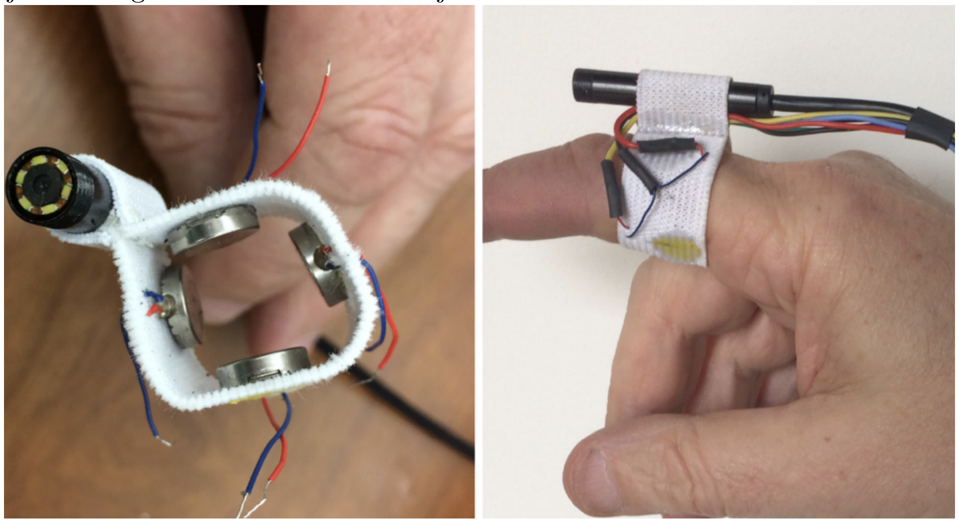
Figure 25.7.: An example of a realization of a mechanical exoskeleton for force feedback in VR and AR applications [299]. The overall mechanical exoskeleton design is shown to the left and the force feedback unit is shown to the right.

Paper Example 25.5.1 : Haptic Guidance for Users Suffering From Vision Impairments

Computer displays do not need to represent digital objects, they could also represent objects in our surroundings. Augmentation of sensory capabilities is valuable for people with sensory impairments, who can be assisted in navigating environments and interacting with objects. *Sonification* encodes geographical or other information to sound, which can be used to assist navigation or situational awareness of blind users. *Vibrotactile displays* have been developed that vibrate according to interesting objects or events in the environment. Such displays can be placed in multiple ways on a human body; solutions have been presented for the waist, the shoe, the wrist etcetera. Each location, though, triggers different challenges for electrical engineering. Each location is also different in terms of people's ability to discriminate haptic stimulation. The underlying reason is that mechanoreceptors are more densely located on certain parts of the body. In this respect, placing such displays on fingers seems appealing.

FingerSight is a haptic guidance system that helps people with vision impairments to locate and reach to objects in peripersonal space [711]. *Peripersonal space* refers to the physical space surrounding our bodies within which we can reach to and grasp objects. While it may appear effortless to grasp an object for a person with no visual impairment, it is effortful for blind people. For example, at the moment of writing this text, the co-author (AO) had the following objects in his peripersonal space: a mobile device, a coffee mug, a water cup, a key, a plate, a knife, a laptop, a chair, a water jug, and a piece of paper.

FingerSight consists of a finger-worn ring that houses a camera and a haptic display (**??**). The display has four evenly spaced tactors that can stimulate the index finger. The camera tracks objects to track the distance to them. Distance is mapped to the features of the haptic stimulation, with the purpose of indicating how far away an object of interest is. As the user moves the hand in the air closer or further from an object, the tactors modulate their actuation frequency accordingly. An experiment showed that very little training was needed. Users could quickly learn to discriminate haptic feedback in main directions. Such guidance can help users explore near-by objects and guide the hand to an object of interest.



25.5.3. Haptic textures

Look at a physical surface close to it and touch it, for example, with a finger pad. Close your eyes while doing the following: Then move the fingerpad on the surface slowly: Can you sense the texture of the object? What if you increase the velocity?

Fingertips and other parts of the hand have high densities of mechanoreceptors. This permits both spatial (where) and object (what) sensing. Research has found that much information can be conveyed with temporal information alone [454]. This has opened the door to convey information via *haptic textures*: sensations of texture generated when moving a finger on a matrix of actuators.

A high quality tactile matrix can generate haptic gradients that can closely mimic the sensation of brushing against an object. The force of each actuator is calibrated according to the (real) surface normal it is supposed to mimic. This allows generating sensations of slopes and bumps. Their response can further be changed according to the velocity with which the finger is moved. With a high resolution haptic display, global and local areas of the display can be used to convey information. For example, the whole display could resemble a 'sandy' texture, but locally there could be peaks and valleys signifying information.

25.6. Emerging Forms of Displays

This section reviews some ways in which researchers have worked toward the ultimate display, as envisioned by Sutherland [788] in 1965. Sutherland speculated on what the best possible display would be. In the paper, he described the so-called Ultimate Display; the final paragraphs read as follows.

> The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal. With appropriate programming such a display could literally be the Wonderland into which Alice walked.

Since then, HCI researchers have pursued different variants of this display. Next, we present three of these variants.

The first departs from a simple, yet profound idea: What if pixels were not limited to communication via emission of light? Most of this chapter has focused on visual displays. In contrast, *shape-changing displays* use physical actuation to encode additional information into pixels. Users can touch, push, and explore this information with their hands.

A prominent example is *inFORM*, developed by Follmer et al. [246]. It was inspired by the concept of affordance: the perception of action potentials in an object. The system, shown in Figure 25.8 uses 30x30 motorized polystyrene pins arranged in a 381x381 mm lattice. Pins are attached to push-pull rods that allow the extension of a pin 100 mm

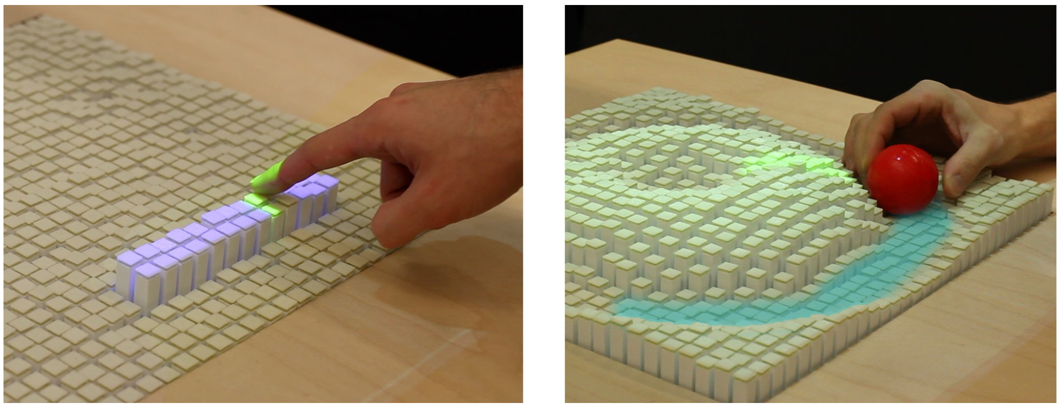


Figure 25.8.: inFORM is a shape-changing display. It enables physical affordances via physical actuation of pixels [246]. The technique allows haptic perception of information encoded a pixel. A pixel can communicate via information encoded in (1) color, (2) height, and (3) dynamic response to pressing (e.g., stiffness).

from the surface. Motorized slide potentiometers are used to control the rods, producing a force up to 1.08 N. PID (proportional-integral-derivative) is used for real-time control of haptic feedback upon pressing. In the prototype, a ceiling-mounted projector was used to project color on pins.

The prototype allows for exciting new interaction techniques (see Chapter 26) that are impossible with regular touch displays. *Buttons* can be formed by raising pins and responding with a button-like stiffness to pressing. *Touch tracks* are lines or curves formed by adjacent pins, which the user can touch at different locations or slide over. *Handles* are pins that can be pinched and pulled up or down in own dimensions. Dynamic widgets can be created that change as a response to a user's press. For example, a triangular Play button can change to a rectangular Stop button in a media player. Pins can also react to *hovering* to create what the authors call dynamic affordances. With a ceiling-mounted camera for hand tracking, pins can lift up those areas of a display that are interactive. Because the pins have sensors, other objects also be placed on the display. For example, a ball can be rolled on the display, the display reacting to it to create a path for it. An object placed on the surface can be rotated to create a knob-like experience.

Most typical displays have been rigid, tablet-sized, or display-sized, and of particular materials (e.g., glass, plastic). Much research on displays have attempted to move beyond these sizes and materials.

Finally, *digital fabrication* concerns the computer-supported process of designing and creating physical objects. It has revolutionized many areas of manufacture and has also been influential in the field of HCI. One reason for this influence is that it allows some parts of Sutherland's vision to be fully realized: The creation, replication, and manipulation of physical objects.

Zoran and Paradiso [911] presented early work on such digital fabrication. They aimed

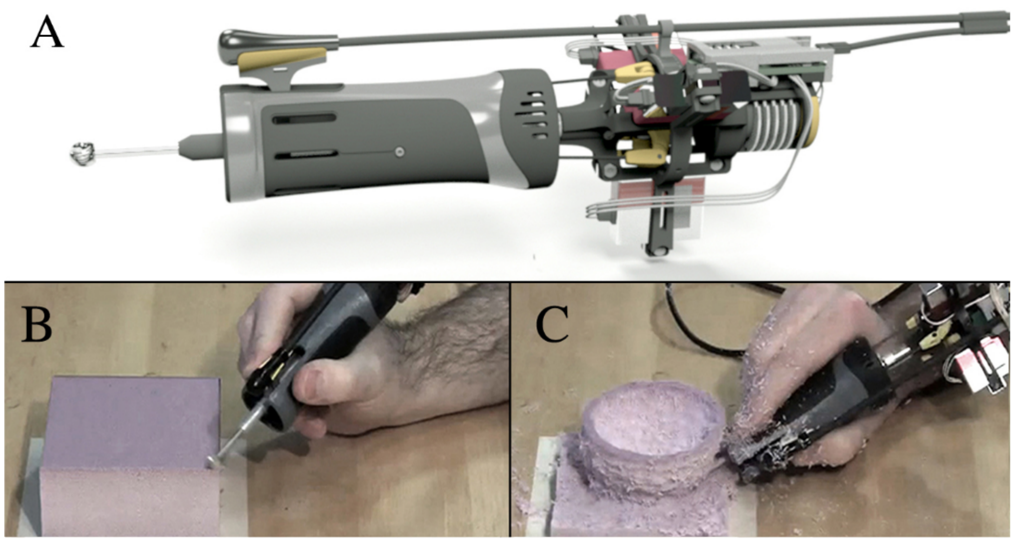


Figure 25.9.: FreeD, an early tool for fabrication [911].

to create a device that bridged on the one hand digital models and traditional craft on the other. This resulted in the tool shown in Figure 25.9. At the top (A) is shown a handheld milling tool that allows the user to cut into different materials. The tool is also motion tracked so that the location of its tip is known. This means that when the user removes material from an object to be created (B), the tool can change spindle speed or move the shaft to alert the user if they are violating the constraints of a pre-selected 3D digital model. In that way (C), the user is supported in adhering, if they wish, to the constraints of the model, while being allowed craftmanship and freedom of expression. In this case, we consider digital fabrication a form of display or output.

Summary

- Displays turn digital information into physical phenomena that users should be able to perceive accurately.
- A key consideration is their match to the human visual system. Any change in a display, be it visual or otherwise, may not be perceived or perceived proportionally by the user.
- Displays are not limited to visual presentation of information and they can be implemented via different technical principles.
- Emerging displays combine displays and sensing.

Exercises

1. Dot matrix displays. What cannot you display with these?
2. Designing a visual display. Consider having to design a real-time updated scatterplot display of a two-dimensional dataset for a VR environment. 1) Which visual channels would you use? 2) How would you encode the variables visually? 3) What are the pros and cons of a VR display over a regular desktop monitor?
3. Haptic interfaces. Following the concepts given in this chapter, propose how to design a haptic notification (e.g., for an incoming message).
4. Sensory modalities. Compare the auditory and the tactile sense as modalities for displaying geographic data.
5. Consider small point lights (e.g., LEDs), used in kitchen appliances, cameras, watches, and many other interactive systems. They can vary in intensity over time. What are the possible ways you could use them to encode information? What are the likely perceptual limits for people trying to decode the information? When done, have a look at Harrison et al. [310] to see some intriguing design possibilities.

26. Interaction Techniques

Although the term itself may not be familiar, we use *interaction techniques* all the time. If you open an application menu on a computer, what you will see are hotkeys (or shortcuts) associated with commands; In the 'Edit' menu, you may, for example, see 'Ctrl-X' for cutting and 'Ctrl-V' for copying. Alternatively, you can click on the command with the mouse. When you hover over the item in the menu, it is highlighted by changing the background color. Using hotkeys or the mouse to select a command differs in multiple ways. Using hotkeys is fast, but requires learning. Selection with a pointer is slower because it requires visual search and pointing.

All basic interactive tasks that we carry out with computers are facilitated by interaction techniques such as pointing, scrolling, menu, entering text, and navigating. Some familiar techniques that we discuss in this chapter include scrolling, word autocompletion, panning, and zooming, and gain functions in pointing. At a more general level,

> an interaction technique is a computation that couples input and output, to support elementary interactive tasks.

Figure 26.1 shows some examples of interaction techniques; the side box below explains the Bubble Cursor, an interaction technique that expands a cursor's selection area to always hit the nearest target.

What are the elementary tasks that users need to solve? In the bulk of this chapter, we discuss the following. The literature contains many others (see for instance the work of Foley et al. [245]), but these four are essential.

- Moving the active area or cursor in the user interface, often called pointing techniques.
- Selecting or manipulating an object; this includes menu techniques
- Entering numbers and text, often called text entry techniques.
- Changing which part of an information space the user sees, often called camera control or navigation techniques.

To help users solve these elementary tasks, interaction techniques couple input, from an input device (see Chapter 24), to output, shown to the user on displays (see Chapter 25). Occasionally, that coupling is simple: A hotkey couples key combinations where a modifier key and a letter are pressed to executing a command, with some feedback to the user that the command has been invoked. At other times, that coupling is simple. For instance, most operating systems use some form of mouse acceleration, which may involve complicated mappings from movements of a mouse to the movement of the cursor pointer [138].

Coupling distinguishes interaction techniques from other concepts discussed in this Part. *Input methods*, for example, concern primarily input processing, and *input devices* the underlying hardware and software system. *User interfaces*, on the other hand, are much larger wholes. They may involve multiple interaction techniques and other interactive features designed to support several user tasks. Except for simpler mobile games, one interaction technique rarely suffices as a user interface to an application.

Next, let us look at what matters to interaction techniques.

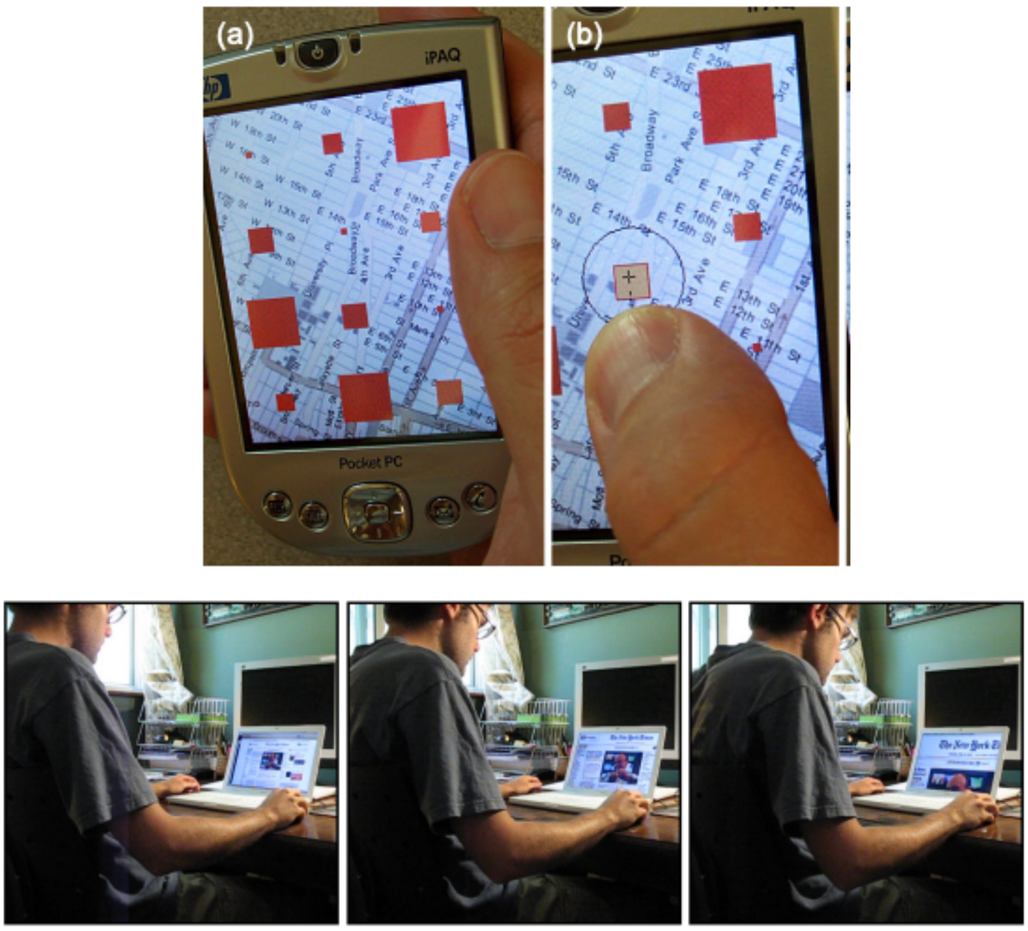


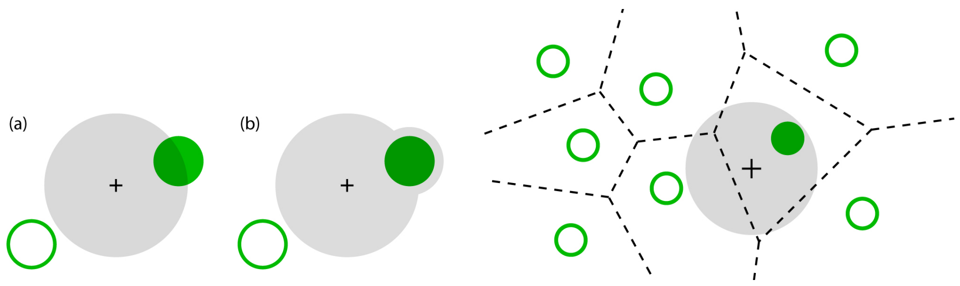
Figure 26.1.: While we commonly use interaction techniques like scrolling, panning, and zooming, HCI innovates and studies alternative designs and their pros and cons. Two examples: Top: a way of shifting icons, red squares, so that they are not obscured by a user's finger [841]. The position of the squares is shifted from so that the user can see them. Bottom: An interaction technique where leaning forward zoom the content on a display [308].

Paper Example 26.0.1 : The end of misclicks?

Bubble Cursor is an interaction technique to select targets with a pointing device [289]. It facilitates the selection of targets when using an indirect pointing device, such as a mouse. The intuition is easy to get: Imagine pointing toward a small target. If the target has no neighboring targets, why not make its selection area much larger? This would make it easier to select.

Its computations consist of three steps. First, the Bubble Cursor computes a geometric shape called a Voronoi diagram for targets on the display. Given the targets, a Voronoi diagram partitions the display so that every pixel on the display is mapped to exactly one target. This defines *selection regions* for the cursor. Second, when the cursor is moving, the Bubble Cursor actively changes the selection region; in other words, to which target the cursor is mapped to at any given time. Third, clicks are assigned to the target within the same Voronoi cell as the cursor. The figure shows how moving the cursor would not normally select a target (a), but when using the Bubble Cursor (b), it would be. The third pane shows the corresponding Vornoi diagram.

The authors' hypothesis was that the technique benefits the selection of smaller targets because the empty space around them can be exploited. In particular, the Bubble Cursor expands their effective selection area, possibly putting an end to misclicks. This hypothesis was evaluated in an empirical study in which users were asked to select targets as quickly as possible. They found that the benefit is large and practically relevant: The average selection time was 1.25 s for the regular point cursor and 0.9 s for the Bubble Cursor. The downside of the technique is that it changes the familiar look and feel of pointing and may obstruct graphics or text underneath the bubble. However, with an increasing density of distractors, this benefit vanishes. This finding shows that empirical studies have a critical role in helping to identify the operating range in which a technique is useful.



26.1. Objectives for Interaction Techniques

Why do we need interaction techniques? The two main purposes of interaction techniques are to improve and enable interactions. First, some interaction techniques improve user performance in some conditions of interest. The goal is to find a technique that suits the user group or the context. Gesturing – shapes and directed swipes drawn on a touch display – offers an alternative to tapping in text entry. Similarly to hotkeys, gestures require some time to learn; however, they may speed up text editing. The stylus, a digital pen available for some tablets and smartphones, is another example. However, it offers superior tracking and drawing performance at the expense of mobility: Using a stylus requires adequate posture and support. Some design objectives commonly considered in the research of interaction techniques are given in Table 26.1.

Second, interaction techniques enable interaction in circumstances that would normally be challenging or even impossible. Research on enabling techniques often attempts to exploit alternative *sensory modalities*, such as gaze or haptics. In *Augmentative and Alternative Communication* (AAC), interaction techniques are developed for people with speech and language difficulties. *Gaze-based interaction* refers to a class of interaction techniques developed primarily for users with severe motor disabilities, such as cerebral palsy. An example is *dwell-based selection*: Here, a target can be selected by fixating on it for a long enough time. A threshold is set for continuous fixation on a target, after which it is triggered. The downside of this technique is *the Midas effect*: gazing at something may inadvertently select it. Hence, setting the threshold incurs a trade-off between the speed of selections on the one hand and false positives (Midas selections) on the other.

An interaction technique often exposes a novel *trade-offs* for an interactive task. In the case of hotkeys, for example, the trade-off concerns performance and learnability. Hotkeys enable an alternative *modality* to access commands. Instead of tapping–or point-and-click if you are using a mouse–pressing a key combination launches the command. On the one hand, hotkeys are quicker to access than accessing commands by pointing. If you know the hotkeys of the commands that you use the most frequently, you can be significantly faster than someone who does not. An expert user of professional photo editing software has tens of hotkeys. On the other hand, learning key combinations takes considerable time. Several hours of practice are needed to adopt only ten hotkeys. An example of tradeffs in design is given in the side box.

Research on interaction techniques involves three main types of efforts: (1) innovation of novel input–output couplings, (2) engineering work to enable, implement, and optimize technical enablers of techniques, and (3) empirical research on how interaction techniques work in practice. Advances in software and hardware have opened up many opportunities for innovation. We have new methods for sensing human movement, from eye-tracking to EEG. We have more responsive, higher resolution display technology for a variety of circumstances from cars (HUDs) to virtual reality. We have new and more accurate machine learning methods and compute resources, which allow us to infer, reason, and plan in real time. The prevalence of possibilities has created a dilemma: Almost any interaction technique could be created, but which ones *should* be created? Answering this question requires commitment to user-centric design processes.

Consequently, research is a multidisciplinary effort drawing contributions from computer science (e.g., machine learning techniques for inferring what users intend), electrical engineering (e.g., sensing methods suitable for gaze-based control) and interaction design (e.g., designing gestures for text entry on mobile devices). Design decisions can also be informed by psychology, especially research on motor control and perception. The task related to the interaction technique can be analyzed and the corresponding motor control

Design objective	**Operationalization**
Performance	Speed and accuracy in representative tasks
Experience	Self-reported experience of flow, mastery, control, or similar quality
Learnability	Rate of learning; Time it takes to achieve a desired level of skill
Mobility	Suitability for conditions in which users are walking, multitasking, or otherwise encumbered
Ergonomics	Physiological and experienced comfort in representative tasks
Accessibility	Suitability for users with different abilities

Table 26.1.: Common design objectives considered for interaction techniques

task, such as pointing or steering, can be identified. A corresponding theory or model is used to engineer the technique or assess potential design solutions. Techniques can also be empirically evaluated in an experimental setup where users need to perform that task. Often, the performance achieved is compared to *baseline design* to assess it. Baselines are often previously known state-of-the-art techniques or designs popular in everyday use. Without a baseline, it may be difficult to demonstrate that a desirable trade-off has been achieved.

In the rest of this chapter, we first review key areas of research, starting with pointing techniques. Finally, we look at two important perspectives on interaction techniques: control and learnability.

Paper Example 26.1.1 : Tradeoffs in the design of interaction techniques

The design of interaction techniques involves multiple objectives. Hotkeys are a point in case. Grossman et al. [290] compared several techniques for leaning hotkeys (Figure 26.2):

1. Traditional: The hotkey is presented next to the command label as in desktop applications.
2. Fading-out hotkey: The hotkey label stays visible for a short time after selection and then fades out.
3. Hotkey menu replacement: The command label is replaced with the hotkey label when the user hovers over it.
4. Audio feedback: The command label and the hotkey are both read aloud by the system after selection.
5. System delay: the user is forced to look at the hotkey label of a clicked item for a few seconds.
6. Disabled menu items: Command labels are grayed out, while shortcuts are more clearly visible.
7. Blinking hotkey: When the menu closes, the hotkey stays visible and blinks for a short duration
8. Following hotkey: when an item is clicked, the menu closes but the hotkey stays visible and follows the mouse cursor for a while.
9. Visual feedback: combines blinking, following, and replacement techniques.

The set of techniques may look arbitrary at first, but they actually combine three design principles: Cost, Feedback, and Memory. First, by manipulating the time cost associated with hotkeys, one can incentivize users to consider them (cost-based techniques); In Chapter 21 we discuss why changing costs and benefits can nudge users to adopt different sensorimotor strategies. Second, hotkeys are often not noticed. Making them more prominent may promote the recognition of them (feedback-based techniques). In Chapter 3 we looked at visual attention and saliency. Third, using different sensory modalities can help encode and recall hotkeys better (memory-based techniques).

A controlled, isolated task was used to compare these techniques. In the task, a command label is shown, and the user must select it from the menu as quickly as possible. The results were surprising. Audio feedback and Disabled menu techniques worked the best, both increasing the use of hotkeys by more than 100 % and reducing task completion time, a staggering improvement! In the Traditional condition, only 30 % of the users used the hotkeys at all, and this doubled with these two best techniques. Users also started to use hotkeys earlier in the experiment. However, there are caveats. Audio feedback works by continuously reminding users about hotkeys, perhaps to the point they give up and rather start using them than continue listening to them. Disabled menu items, similarly, force users to pay attention to the hotkeys. Visual feedback, to the surprise of the authors, did not help transitioning to hotkeys, even if this was combining other techniques. Moreover, users did not appreciate it. The authors speculated that this is because users must focus on a visual main task and could not divide attention to the cues. Also the Delayed technique was negatively perceived.

26.2. Pointing techniques

A pointing facilitation technique maps changes registered in an input device to the movement of a cursor or the presentation of potential targets on display. Two-dimensional pointing is familiar to us from desktop interaction (WIMP paradigm, see Chapter 28). We find one-dimensional pointing in *scrolling*. Here, the end-effector must be brought on top of a region that corresponds to a page or location in the document. Three-dimensional selection occurs in augmented and virtual reality applications.

Pointing facilitation techniques can be further distinguished into three classes:

1. Input transfer functions: Techniques that map changes in input signal to cursor movement. Example: CD gain functions;

2. Display transfer functions: Techniques that change the presentation of potential targets to facilitate their selection. BubbleCursor, discussed above, is an example of a class of techniques that change the selection region of elements;

An understanding of pointing facilitation starts from an understanding pointing. As we saw in Chapter 4, pointing is an aimed movement. An end-effector, such as a cursor, is brought to refer to an object of interest. In the case of HCI, pointing often includes clicking or some other way of selecting an area under the end-effector. The understanding of this elementary interaction is based on Fitts' law, given in Chapter 4.

What does this model tell us about pointing techniques? It says that both distance and width have an effect on MT. Hence, if the interaction technique can change those, it can decrease MT. For example, BubbleCursor makes W larger and D smaller. However, the two empirical constants may hide some other effects. Some techniques adapt to the user, while others change response dynamically depending for example on speed of motion or what is on the display. a and b are not constants.

26.2.1. Control-to-display (gain) functions

As a case of pointing facilitation techniques, we consider the transfer functions that map movement on a control surface to cursor motion. *Control-to-display (CD) functions* are input transfer functions used in indirect input devices such as mice, joystick, and trackpads. They interaction techniques used by almost every computer user. A CD

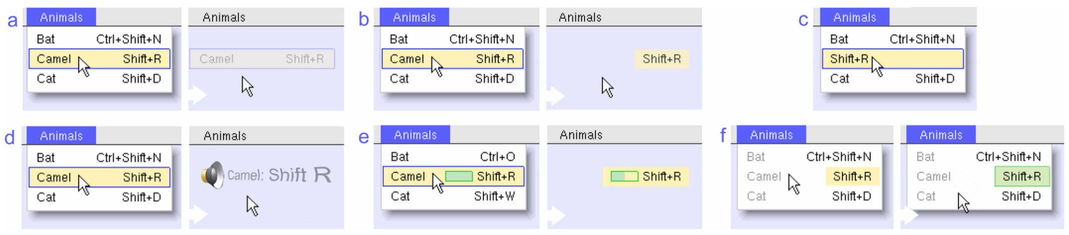


Figure 26.2.: [290] compared five techniques for promoting the use of hotkeys.

function – also known as a gain function – maps changes in *control space* (e.g,. touchpad) to changes in *display space* (e.g., visible cursor). Every indirect device uses some CD function, although in some cases it is simply a linear coefficient.

Technically, a gain function computes a scalar factor (or "gain") from the instantaneous input speed (v_{in}):

$$v_{out} = f_{CD}(v_{in}) \times v_{in}. \tag{26.1}$$

CD gain functions can be as simple as a fixed gain: A constant ratio mediates input and output movement velocities. But how to pick a good constant? Imagine that you are designing a new augmented reality technique for selecting distant targets. Local hand movements (control space) are mapped to movement of warped hand movements shown in distance. Casiez et al. [137] presented a principle to choose a range of gain values: First, minimum gain CD_{min} must allow reaching the most distant targets without clutching (lifting the mouse or finger from the surface), and the maximum gain CD_{max} must allow accessing an individual pixel. Setting a single value is necessarily a compromise between these extrema.

For this reason, most commercially used gain functions are *speed-dependent*: that is, the gain changes as a function of velocity. It is agreed that speed-dependent functions are better, because they can respond to both fast movements, reducing the need for clutching, and to slow movements, needed for more accurate pointing. However, most functions are proprietary – their shape is not disclosed to the public.

Casiez and Roussel [136] reverse-engineered gain functions in existing operating systems, discovering that they share some features, namely (1) monotonic increase at the beginning and (2) comparable maxima. They were distinguished by the minima, continuity, and shape of the functions. The functions are shown in Figure 26.3. Speed-dependent gain functions have been shown to be up to 24% faster than a constant CD gain in an empirical evaluation. Such differences have practical significance, given the high frequency of pointing in human–computer interaction.

Why do such differences emerge? There are four main causes. First, speed-dependent gain functions can dampen excess speeds. Since high-velocity movements cannot be controlled accurately (remember speed–accuracy trade-off in Chapter 4), they can be made more predictable by sloping of the curve to a near-constant speed at that extrema. Second, they can add more granularity to high-precision movements. Third, the different velocity ranges are associated with different muscle groups. Consider very high-precision motions with a mouse: you probably do those with wrist and finger motions. High-speed motions, on the other hand, recruit muscles in the arm and shoulder. Fourth, they can reduce unnecessary clutching.

26.3. Selection and manipulation techniques

Selection and manipulation techniques affect an object once it has been identified, for example, by pointing. The simple case, selection, simply selects an object. This might invoke a command coupled to the object or mark for the object as the argument for a subsequent command. More complex selections, as in the case of menus, have allowed inventive selection techniques that help to effectively select among an array of options. Finally, objects may be manipulated in other ways—resized, moved, or rotated. Many interaction techniques have been developed for these cases. Let us look at these techniques in turn.

26.3.1. Simple selection

Usually, selection in graphical user interfaces is simple. You click the mouse over an object or tap it with your finger. However, sometimes this is difficult; recall the chapter on motor control (see Chapter 4). The target might be small. Your finger may obscure the target. You may be unable to look at the target. You may shake as you attempt to perform the selection. Effective interaction techniques deal with these cases.

For instance, the Shift technique shown in Figure 26.1 offset targets so that the user may see what is below their finger [841]. Another idea is to make that which is selected

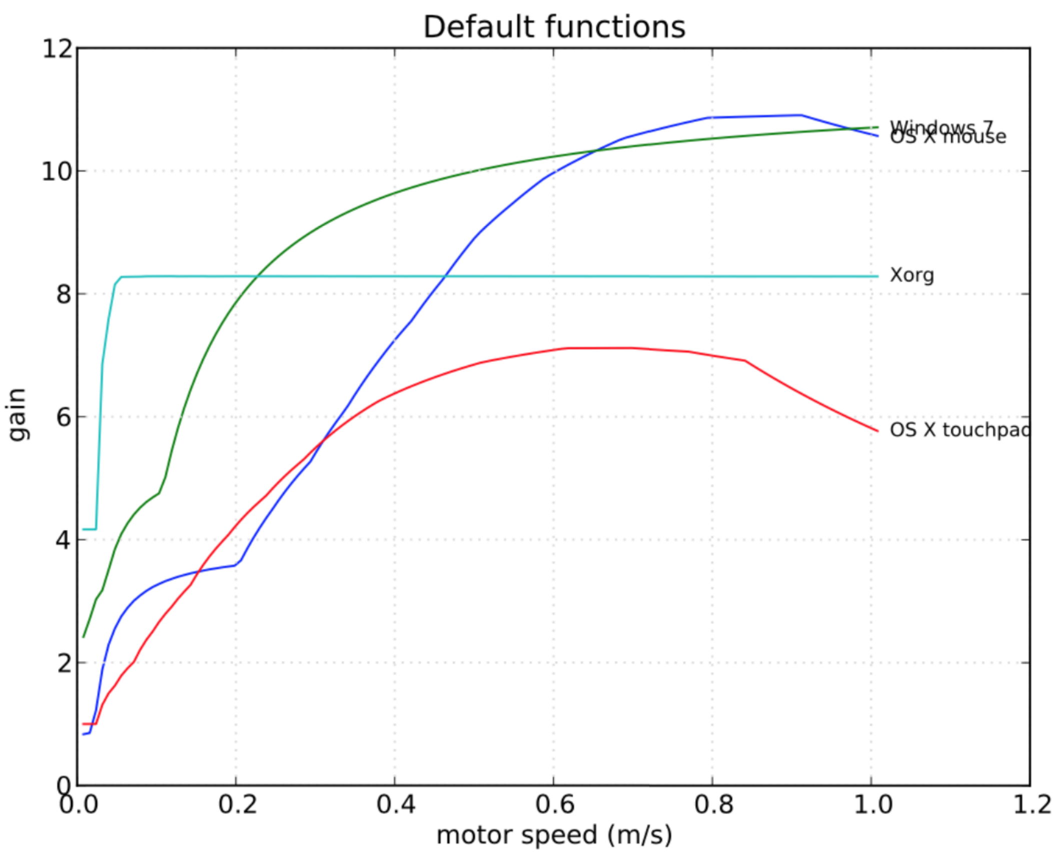


Figure 26.3.: Casiez and Roussel [136] proposed a novel method to reverse engineer CD functions and found large and surprising differences among modern operating systems.

sticky [895]. A sticky icon, for instance, reduces the cursor's gain ration when the cursor is over the icon. This makes it easier to stop over the icon and, therefore, it is easier to select it. Worden et al. [895] showed that such icons make selections faster, in particular for smaller icons, across both younger participants (mean age of 23) and older participants (mean age of 70).

Another idea is to eliminate the need for clicking as we normally use it. In *crossing*, the user crosses the object with the mouse or the finger, rather than clicking on it. In *dwell-time selection*, the user hovers the object to be selected and after a certain time over the object (the *dwell*), the object is selected.

In this section, it is assumed that the selection is binary—either you select an object, or you do not select it. But that need not be the case. Ramos et al. [668] implemented and evaluated pressure widgets, user interface elements that allow users to select with a degree, indicated by their pressure.

These and many other works on simple selection techniques show that even for something as simple as a button click, researchers in interaction techniques have been remarkably inventive.

26.3.2. Menu selection

An interactive menu system organizes a collection of items on display so that the user can explore the menu and select the desired items. Graphical menu systems present menu items visually. Menu systems often use hierarchical organization, where items are placed under submenus. Another defining feature is semantics: the options have labels that have semantics – they mean something. Associations between options are exploited in the organization of the menu. Items that belong together are often put into the same unit, for example, a submenu. Design can exploit this in different ways. Menus are often organized alphabetically, numerically, or group semantically.

Typical to larger menus is that they utilize *transient visualizations*. That is, the menu, or a part of it, is temporarily displayed, and can be dismissed. For example, when you click on the title of a menu on a workstation application, the submenu appears below. But when you move the cursor away from the *activation area*, it disappears. Two menu techniques – a morphing menu and a split menu – are shown in Figure 26.4.

Pie Menus

While hierarchical linear pull-down menus dominate in most modern graphical user interfaces, they are not only means of representing a menu structure. A *pie menu* [121] presents menu as a circle subdivided into circular sectors associated with commands or further menu structures (Figure 26.5). The user typically triggers the pie menu using a button press (such as pressing down the left mouse button). When the pie menu is invoked, it appears in the center of the pie menu. The user can now perform a selection by sliding the cursor towards a circular sector. Upon a selection, the circular sector is highlighted to visualize its selected state. If the user lifts the trigger button (such as a left mouse button) the command associated with the circular sector is issued by the system.

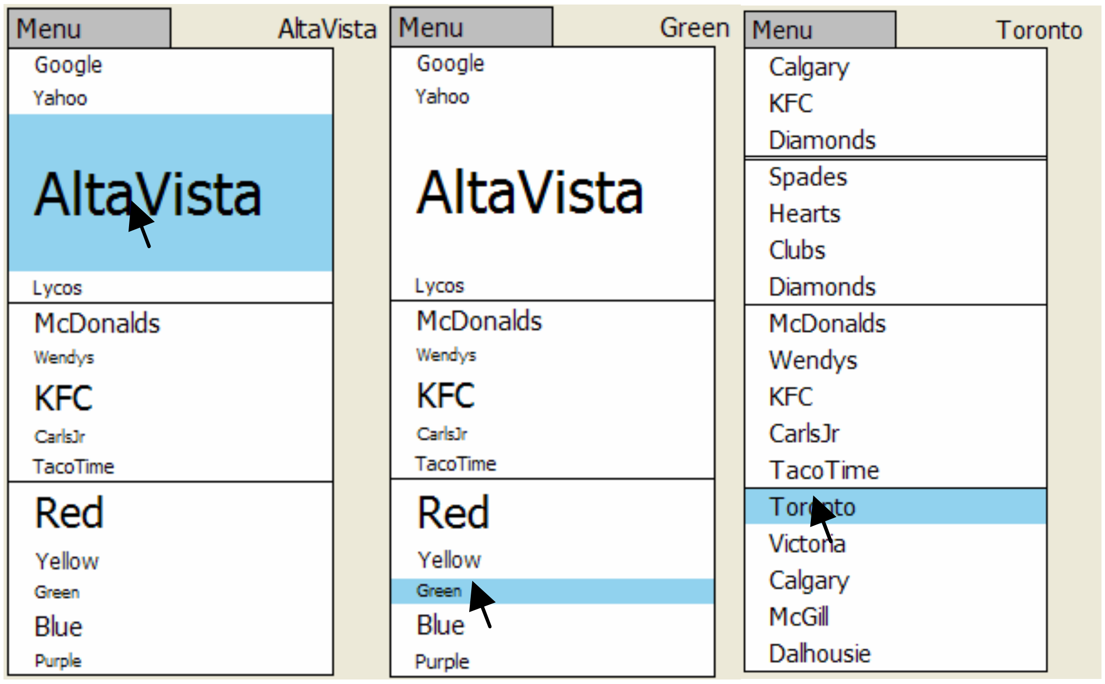


Figure 26.4.: Two morphing menus and a split menu.

Pie menus are not as popular as hierarchical linear pull-down menus. However, they have been used in, for example, games and computer-aided design applications.

Pie menus can be hierarchical because a circular sector can be associated with either a command or a menu structure. If the selected circular sector is associated with a menu structure, then a selection will invoke a second pie menu revealing this menu structure. In this way, pie menus allow the user to navigate a hierarchical menu structure in a similar vein as a hierarchical linear pull-down menu.

An advantage of both hierarchical linear pull-down menus and hierarchical pie menus is that they are easy for a novice user to use, as they allow the novice user to browse all available commands in the application and select them. Assuming the menu hierarchy is reasonably logical, such navigation, while sometimes time-consuming, should limit frustration.

However, a disadvantage is the low performance ceiling. Since navigating pull-down menus and pie menus requires visually-guided closed-loop motion, it is not possible for users to reach a high performance even when they know precisely how to navigate to a desired item. This is because visually-guided closed-loop motion imposes a low performance ceiling, as discussed in Chapter 23. As an alternative, pull-down menus often use keyboard shortcuts. However, such keyboard shortcuts force a conscious shift from one mode of interaction (visually-guided menu navigation) to another (direct recall by triggering a key combination).

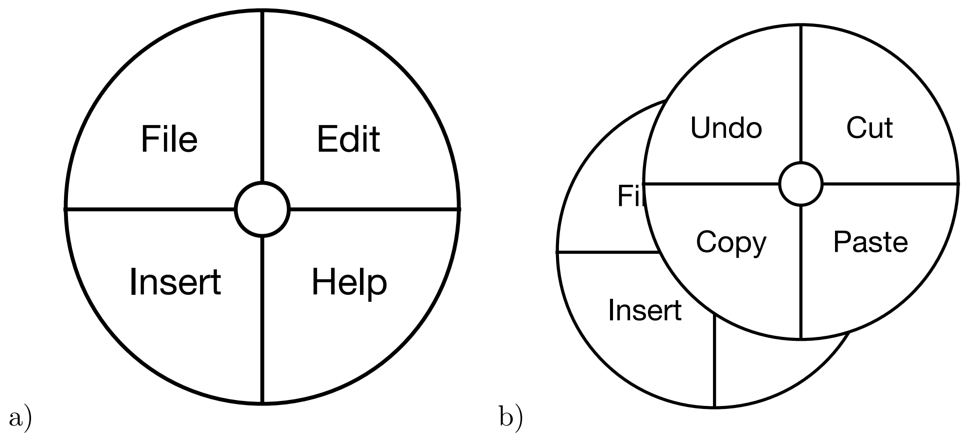


Figure 26.5.: a) An example of a first-level pie menu. b) An example of a second-level pie menu.

Marking Menus

A *marking menu* [437] is a refinement of a pie menu that supports a *continuous* transition from novice to expert behavior. A marking menu is an interaction technique that couples an incremental angular 2D single-stroke gesture recognizer with a pie menu (Figure 26.6).

The user triggers the marking menu similarly to a pie menu. However, unlike a pie menu, the marking menu does not reveal a pie menu immediately. Instead, there is a brief timeout period before the pie menu appears. After the timeout, the marking menu behaves like a pie menu. However, if the user performs a 2D gesture rapidly before the timeout, the pie menu will not be shown.

The central idea in the marking menu is to ensure that the trajectory for invoking the command, either navigating to the menu item or performing a 2D gesture, is identical. This means that during use, the user learns the motor pattern of the command by consolidating the motor memory. Importantly, this process is both continuous and unconscious. This allows marking menus to provide users with a seamless novice to expert transition.

There are several details in the implementation of marking menus which may vary depending on specific use-cases or designer insights. For instance, the timeout is a controllable design parameter that can be set in various ways. If the timeout is long, users are further "pushed" to invoke commands by 2D gestures. On the other hand, if the timeout is low, users are encouraged to navigate to the pie menu. It is also possible to eliminate the timeout by analyzing the user's movement dynamics. If the user's movement dynamics suggests open-loop gesturing, then the system can automatically turn off the pie menu and transition to 2D gesturing mode.

Some implementations of marking menus use a sophisticated 2D gesture recognizer that allows the user to pause the gesture to trigger the pie menu (at its current selection level) at any stage. Traditional marking menus implement this using a recognizer that focuses on analyzing movement angles. Another design consideration is the number of menu items in the pie menu. Typical choices range between 4-8 menu items. The more menu items that are supported within a single pie menu, the more accurate the user has to be in selecting the menu item.

There are two fundamental limitations with marking menus. The first is the limited number of menu items at any level of the menu. The second is the screen real-estate required to support large continuous gestural motions to select menu items that are deeply nested. One solution that has been explored is to operate the marking menu using discrete single stroke straight line segments, *simple marks*, instead of a single continuous selection across the menu hierarchy, *compound marks*. Simple marks have the advantage that they use less screen real estate and may support increasing the number of menu items within the marking menu [906].

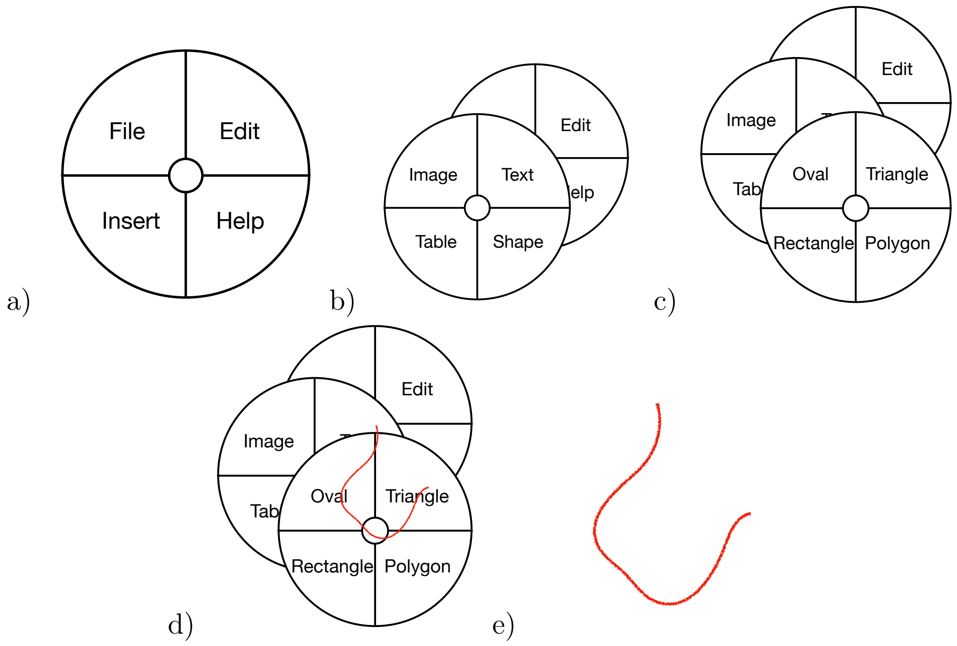


Figure 26.6.: An illustration of a marking menu selection. a) The marking menu is triggered in a similar vein to a pie menu. The user drags the cursor towards the *insert* menu. b) This action triggers a second-level marking menu. The user drags the cursor towards the *Shape* menu. c) This triggers a third-level marking menu. The user triggers the *Triangle* menu item. d) The entire selection motion for the user triggering 1) the marking menu; 2) the second-level *Insert* menu; 3) the third-level *Shape* menu; and 4) the selection of the *Triangle* menu item is illustrated as a red trace. e) This trace forms a 2D gesture which can be directly articulated by the user without invoking the marking menu. Critically, the selection motion for *navigating* to the *Insert Triangle* command is identical to the selection motion for *gesturing* the command *Insert Triangle*. By repeat selection of the same command, the user gradually learns how to recall the command using a rapid gesture rather than slow visually-guided menu navigation.

Paper Example 26.3.1 : A model of user performance in menus

From a motor control perspective, menu selection is a *compound task*; that is, it consists of several subtasks. The *Search-Decide-Point* (SDP) is a mathematical model that illustrates this well [156]. The model is a linear regression model similar to the Keystroke Level Model (see **??**). It predicts selection time in a menu as a sum of time spent in three subtasks:

- *Search*, the time to localize an item, increases linearly with the number of items in the menu;
- *Decide* is the time to decide from among items given the entropy determined by the frequencies of previous selections and given by the Hick-Hyman law;
- *Point*: Pointing is based on Fitts' law and predicts that items closer to the top are faster to select.

Search and Decide subtasks are affected by the amount of experience that the user has with the menu, in particular the number of repetitions of an item. With practice, performance changes from being dominated by search (linear) to decision (logarithmic). The expertise factor e controls this.

Considering a linear menu as the case (**??**), average selection time for a menu with n items is the sum of selection times of each item i weighted by their selection probability:

$$T = \sum_{i=1}^{n} p_i T_i \tag{26.2}$$

Here, selection time T_i of item i is given by

$$T_i = T_{dsi} + T_{pi} \tag{26.3}$$

Pointing time T_{pi} is governed by Fitts' law, such that movement is assumed to start from the top of the menu. Decision/Search time T_{dsi} is linear interpolation between decision and visual search:

$$T_{dsi} = (1 - e_i)T_{vsi} + e_i T_{hhi} \tag{26.4}$$

where $e_i[0..1]$ is the expertise factor $e_i = 1 - 1/t - i$ where t_i is the number of repetitions with i. As the user becomes more experienced with a menu, there is a shift from serial visual search T_{vsi} to decision among competing elements T_{hhi}.

Finally, visual search time T_{vsi} is assumed to be linear function of the number of n:

$$T_{vsi} = b_{vs} n + a_{vs} \tag{26.5}$$

Hick-Hyman decision time T_{hhi} is dependent on the *entropy* of the item $H_i = log_2(1/p_i)$. Here, p_i is the probability of the item in the history of selections, $p_i = t_i/t_t$ where t_t is the number of total selections in the menu. Now, T_{hhi} is given by:

$$T_{hhi} = b_{hh} H_i + a_{hh} \tag{26.6}$$

where a_{hh} and b_{hh} are empirically determined constraints.

The model has been validated for a few different types of menus, including linear, morphing, split, and rectangular menus. Figure 26.4 shows *a morphing menu* and a split menu. In a morphing menu, more frequently used items are enlarged. In *a split menu*, a special section of recently or frequently used items is created at the top of the menu. Morphing menu is modeled by changing the width of a frequent target. Split menu is modeled with two models, one for the split region, the other for the standard region. Empirical study confirmed the prediction of the model that frequency-based split menu is the fastest.

26.3.3. Manipulation of Objects

In addition to selection, users may want to manipulate an object that has been pointed to in numerous other ways. For instance, the use of an expansion or contraction of a finger spread—a *pinch* gesture—is widely used in touch interfaces to manipulate objects, for instance to scale them. While the pinch gesture have been known since at least the early 1990s [51], its use for scaling and zooming is surprising complex. One needs to consider the following questions: 1) How to detect pinching? Wilson [876] contributed an algorithm that could detect pinch gestures from a camera feed robustly; 2) How to deal with differences in hand sizes?; 3) Are people equally good at rotation at different angles? Other manipulation techniques concerns rotation , moving objects, and deleting objects.

26.4. Text entry

Entering text and data is one of the most common tasks carried out with computers. Communications and social media are predominantly text-based. Most information systems and office applications rely on the entering of text and data.

Text entry techniques use computational methods to facilitate the task of entering text. The goal is make text entry more efficient, less effortful, or enable it in circumstances it would not normally be possible. The baseline for text entry is often physical keyboards or numeric keys. Physical keyboards allow people to enter a character at a time and typically have a large activation area. Still, they serve only as a baseline because much research has focused on improving text entry.

Most of the research on this topic has focused on virtual keyboards. *Virtual keyboards*, especially on mobile devices, are small and therefore more susceptible to the inherent inaccuracy and noise in human motor control. A number of reliable findings on smartphone typing have been established that motivate the need for research on intelligent support for text entry [620]:

- Young adults type at around 36 wpm (words per minute) and have an average of 2.3 % uncorrected errors. This level is far off what is achievable with physical keyboards, which is around 52 wpm in a comparable sample [188].
- Typing with one finger is slower than using two fingers. On average, two-thumb typing is superior to other methods of using fingers.
- Errors are costly to correct in comparison to physical keyboards. Users differ in how typo-averse they are. Some users slow down to avoid having to correct errors.
- Typing is not just about motor control, users need to control how they deploy visual attention between the text display (what is typed) and the virtual keyboard.
- Typing while walking or doing other activities slows down typing performance and causes errors.
- Many aging users and users with cognitive, motor, or vision deficiencies have serious difficulties typing with standard keyboards.

Consequently, research on text entry techniques has focused on four main themes:

1. Correcting and completing phrases: Given the observed touch points and characters typed so far, the goal is to correct mistakes or complete the phrase.
2. Offering an alternative unit of entry: From character-to-syllable-, word-, and phrase-level entry.
3. Multimodal and cross-modal entry: Using voice, eye movements, etc. to enter text.
4. Editing text: Facilitating corrections and modifications of text.

26.4.1. Correcting and completing phrases

First, correcting and completing phrases is an active topic of inquiry. Without an intelligent text entry method, when you touch a virtual button on the keyboard of your smartphone, the character that corresponds exactly to the touchpoint on the display is entered. However, because of the small size of the keys, the occlusion problem, and environment-caused noise, touchpoint distributions are very varied. Users often cannot reliably touch a target that is smaller than 10 millimeters. Unless the user slows down, many errors will occur.

Decoding-based text entry techniques use machine learning to try to infer the intended message in the presence of noise in the touchpoints. Popular applications include *autocorrection* and *word completion*. In autocorrection, a machine learning -based decoder tries to infer the message the user may have intended to send, given a list of observed touchpoint coordinates. In word completion, a predictive algorithm tries to predict the next characters, given the ones typed so far.

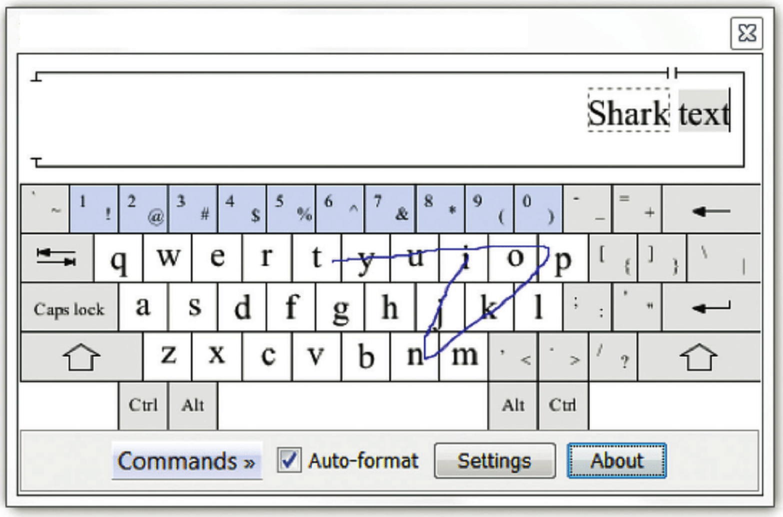
26.4.2. Going beyond entering characters

Second, we can relax the *unit* of entry. Instead of looking at previous characters, we can exploit information in the context of words and sentences. For example, in *gestural input*, the finger does not need to be lifted to individually tap the buttons. Instead, you can keep the finger on the display and sweep a shape that passes through the characters that form the word. The sidebox presents the idea.

Research has recently also looked at *sentence-level entry*, which allows for better exploiting information about preceding words. The two downsides of this approach are, first, the slower computation times and, second, the higher costs of correcting possible errors. The user may need to re-read the whole phrase. If there is an error, it must be detected and corrected by the user.

Paper Example 26.4.1 : Gesture typing with SHARK

Zhai and Kristensson [903] introduced gesture typing, a text entry technique that allows drawing shapes on a touchscreen keyboard. SHARK stands for Shorthand Aided Rapid Keyboarding. The method drew inspiration from handwriting, which however is relatively slow at about 15 words per minute (WPM). In SHARK, instead of entering a word by tapping one letter at a time, a user can draw a shape that goes through the letters that form the word. The technique is argued to support motor learning: after learning a shorthand shape (e.g., ”the”), the user does not have to visually guide the movement of the finger.



26.4.3. Other modalities for text entry

The third research area concerns alternative modalities. Alternative modalities, like speech, can support text entry in circumstances or for people where it could be hard to enter type via touch. However, speech recognition methods are error-prone; moreover, recognition errors *cascade*. That is, an error in an earlier part of a phrase may deteriorate the inference of a word in a latter part. The consequent HCI challenge is to offer efficient ways to edit and correct inferred phrases. For example, *Parakeet* is a demonstrator of continuous speech recognition supported by touch interaction [831]. The user can utter a phrase, which is then displayed on the touch display separated into words. This takes advantage of the *speech recognition hypothesis space*: there is not just one, but several possibilities to infer a word, and making this visible to the user can be beneficial in correcting errors. Parakeet also offers touch interactions that complete speech input. Underneath each word, there are alternative words, which can be tapped to replace an incorrect one in the phrase. Moreover, the user can delete sets of words by drawing through them.

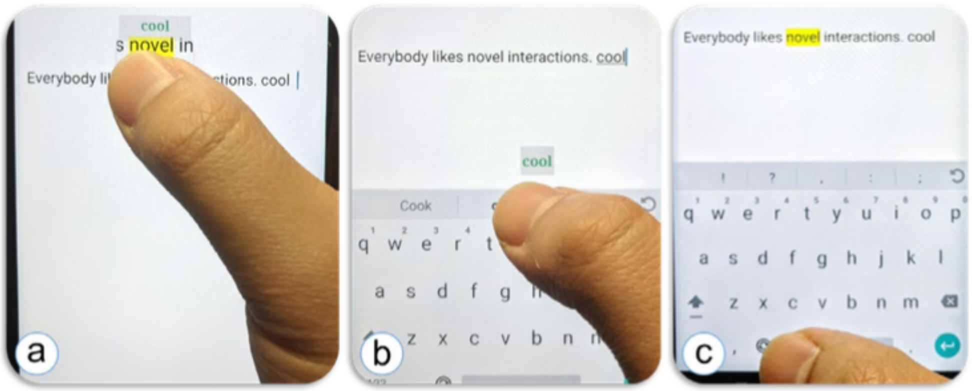

cool
s novel in
Everybody li
tions. cool
a
Everybody likes novel interactions. cool
cool
b
Everybody likes novel interactions. cool
c

Figure 26.7.: *Type, then Correct* presents three correction techniques: (a) Drag-n-drop allows dragging the last word that is typed to replace an erroneous word in the middle of a phrase or to place it in-between (omission error); (b) Drag-n-Throw allows 'flicking' a word from a word suggestion list toward the rough area of a word that needs to be replaced; (c) Magic Key highlights possible error words after a correction that is typed; the user can drag the magic key toward the erroneous word to correct it [905]. These techniques were shown to facilitate error correction over the regular cursor and backspace-based method.

26.4.4. Text editing

The fourth area of research is related to *text editing*. A modern virtual keyboard must support ways to find, replace, copy, and paste text. Changes should be undoable. However, regular text-editing methods are laborious. For example, to edit a character inside a word, a user may need to carefully place the cursor using an offset cursor, because the finger may occlude a part of the word. This is known as the *finger occlusion problem*, also known as the "the fat finger problem". To copy a piece of text, two corners of a widget may need to be carefully dragged to their place. *Type, then Correct* (Figure 26.7) is an example of three editing techniques that exploit machine learning methods: Drag-n-drop, Drag-n-throw, and Magic Key.

To sum up, an outstanding issue with intelligent text entry is the *facilitation–correction cost tradeoff*. The more it tries to facilitate, the higher the speed but also the higher the cost of correcting a possible error. This issue surfaces in particular when phrases contain rare words, like proper nouns, or when an expression combines different languages. On the other hand, if the method offers no facilitation at all, users rely on their own cognitive mechanisms to spot errors, and they are relatively good at that (thus, low correction cost). Emerging large-scale studies suggest that users who use autocorrection are faster than users who do not use it, while users who use word prediction are slower but have fewer errors [620]. In the particular study, perhaps due to the trade-off, users using gestural entry were slower than users not using ITE (intelligent text entry).

26.5. Camera control

In user interfaces, there is often more to see than can fit on the display. For instance, a map be so large that all its details cannot fit on a screen or an immersive 3D world may require the user to move their point-of-view to a different part of the scene. Techniques for camera control helps to change what is shown on the display. The phrase come from computer graphics, where the scene (or content) is often separated from the point of view from which the user sees the scene.

Before looking at some examples, it is worth noting that camera control is similar to pointing tasks (recall Chapter 4). When navigating an information space, positioning the camera at the right place is about pointing to that space. Thus, the familiar terms of throughput may be used for camera control tasks.

26.5.1. Panning and Zooming

A simple example of camera control is panning and zooming. Panning moves the point-of-view across the space. In the 1D case, panning may involve moving a scroll bar to show a certain part of the document; in the 2D case, it may be moving the mouse to the edge of the screen to automatically pan a map.

Zooming changes the magnification of the information shown in the point of view. Zooming is sometimes possible with scrollbars; this changes the size of the knob (the part of the scrollbar that can be grabbed). In the 2D case, zooming is the familiar change of magnification. In *semantic zooming*, this change affect different parts of the information space in different ways. For instance, on a map at low magnification, cities might be represented with generic circles; at high magnification this might change to a detailed outline of the city contour.

User interfaces often offer both panning and zooming. When they can be done at the same time, we say that they are *integral*. On a map, we might be able to zoom by holding a mouse button and move the mouse as we do so. Obviously, integral interaction techniques are more effective when the operation to be done requires simultaneous panning and zooming [373].

One elegant implementation of an integral scrollbar for zooming and panning is called OrtoZoom. One issue in most implementations of the scroll bar is that if the space to scroll is large, the movement in the document coupled to the scroll bar will be coarse. Appert and Fekete [28] developed OrtoZoom, an interaction technique that combines scrolling a 1D data structure (e.g., a document) with a possibility to zoom by moving the mouse in the dimension orthogonal to the scroll bar. Assume that the length of the document is d pixels long and the scroll bar moves up and down. If the user moves the mouse while it is on the scroll bar, each unit scrolled might be correspond to $d/10$ (so that the whole scroll bar may be scrolled in 10 steps. However, if they first move the mouse to the left, each unit scrolled becomes smaller (potentially down to 1). This make scrolling more precise. A related idea is speed-dependent automatic zooming (SDAZ), studied by Igarashi and Hinckley [364], where the viewport is zoomed in or out when scrolling. The level of zoom depends on the information density of the document. The goal is to ensure

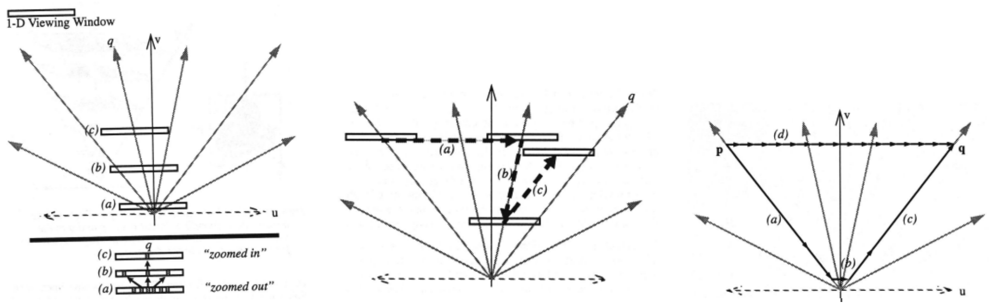


Figure 26.8.: Scale-space diagrams showing some tradeoffs in zooming and panning interfaces; the diagrams were proposed by Furnas and Bederson [260]. To the left is shown an information space with dimensions u and v (depth); the objects in the space are shown as six vectors (q). At any time, there is a viewing window into the information space, shown as a rectangle (a,b,c); at the bottom of the figure is shown which objects that are visible at which magnification. In the middle is shown pure panning (a), pure zooming (b), and integral zooming and panning (c). To the right is shown how panning can be cumbersome. Navigating from p to q can be done as pure panning along the path d. It can also be done as zooming out (along a), panning (b), and zooming in (c).

that the user's visual attention is provided with sufficient time to visually inspect the contents of a viewport.

Both OrtoZoom and SDAZ shows a classic trade-off in panning and zooming: if one pans at the wrong level of magnification, one either risks overshooting the target (because each pan step is large) or having to spend a lot of time panning. However, changing magnification levels is tricky. Figure 26.8 shows the intuition using scale-space diagrams [260]. Much work on zooming and panning seeks to remedy these problems.

A final issue in panning and zooming is to ensure that people can see points of interest and where to go. This, again, is related to magnification level. If you are zoomed in too much, you may see the relevant details but not all the information; if you are zoomed out, you may obtain an overview of the space but not see the details. Jul and Furnas [391] memorably talked about *desert fog*, the situation where you need to reach a target but has no navigational cues where it is. If you zoom in too much, this may be the situation. Paper Example 26.5.1 shows one interaction technique that helps remedy this situation.

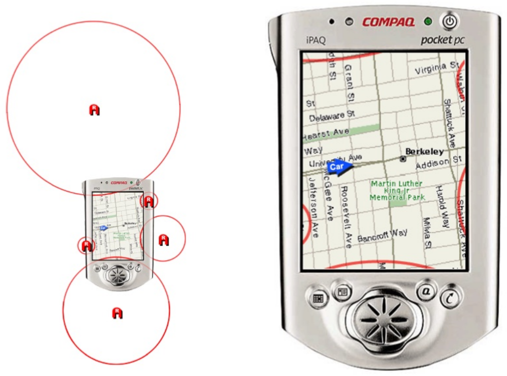
Figure 26.9.: Challenges in 3D navigation encompasses (a) high-degree of freedom in movement and (b) challenges in understanding the landmarks and building up a spatial understanding of space.

Paper Example 26.5.1 : Visualizing off-screen targets

One problem with a small viewport is that it may not be possible to display all relevant information. For example, the figure below to the left shows five target locations on a map. When the user zooms, shown in the figure below to the right, the target locations are no longer visible as the viewport is too small to show them.

One solution to this is a halo visualization [52]. The idea is to center each target location within a circle and let the circumference of the circle indicate distance. In addition, arcs belonging to target locations further away are shown to be more translucent, providing a secondary channel of information about the distance of an off-screen target. The halo visualization concept is illustrated in the figure below to the right.

Unlike, for example, using arrows to indicate off-screen targets, the halo visualization does not require additional annotations to indicate the distance to off-screen targets.



26.5.2. Navigating 3D

A particular challenge for camera control concerns navigation of 3D. There are two reasons behind this challenge (see Figure 26.9). The first is that 3D navigation has three degrees of freedom, sometimes referred to as pitch, yaw, and roll. This is complicated to control for users. The second is that as users navigate in 3D, they build up an understanding of the objects in the space. The way particular interaction techniques help do that differs.

The many degrees of freedom for 3D navigation is particularly challenges if the input device that controls the navigation is not 3D, but for instance a mouse. The many degrees of freedom for 3D navigation is A challenge in 3D navigation is that it may be quite cumbersome when large distances are to be covered. This is similar to the case for panning. The second challenge mentioned above concerned how interaction techniques affect how users build up an understanding of a spatial environment.

26.6. Two perspectives: Control and Learnability

Surprisingly, advanced interaction techniques are not common in everyday use of computers. We mainly use them in a handful of tasks: when scrolling, selecting targets, or entering text. It has turned out to be challenging to create new interaction techniques that users would adopt. To conclude the chapter, we discuss two key challenges in their design: control and learnability.

26.6.1. Control

Control theory (see Chapter 17) can illuminate the dynamic relationships that input actions and events on display have, and what the consequent challenges are to the user. A minimal analysis starts with the following elements:

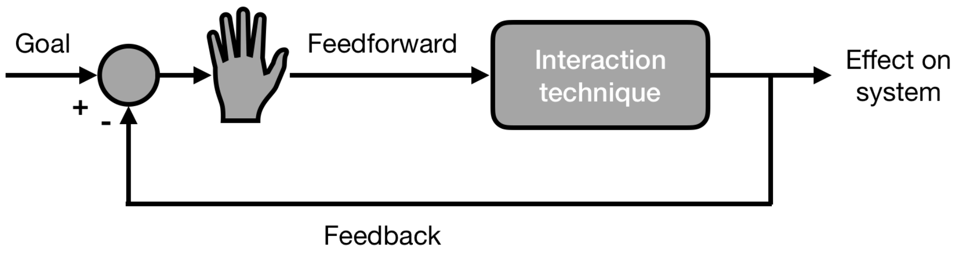
- **Goal**: The user has a goal in mind, a state that the computer should be driven to, such as selecting a particular target on display;
- **Feedforward**: The user sends a control signal via the input sensor to change the state;
- **Transfer function**: A program that maps changes in the feedforward signal to changes on the display;
- **Feedback**: The state-change is now visible on display;
- **Comparator**: The feedback is compared against the goal state in order to determine the new feedforward signal.

In the control model shown in Figure 26.10a, the user chooses a feedforward signal as a linear function of how far the end-effector is from the target. In other words, the user tries to close the gap directly between where the cursor is and where it should be. Assuming that the system is noisy, this will have to be done a few times to hit the target region.

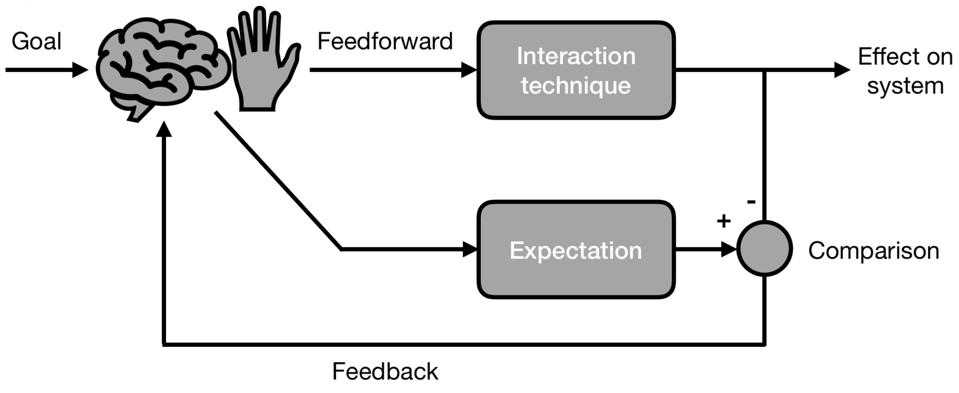
However, this view is overly simplistic. A user deciding what to do next simply by minimizing distance between perceived and desired states would fail. This policy does not account for the non-linearities and dynamics that are inherent in motor control with an interactive system. Consider pointing with a mouse: the muscle control signal (feedforward) that you pick is not linearly determined by the distance that the cursor has to the target (feedback). When you move your hand, the involved biomechanical events are non-linear. Eyes guide the hand, and several muscle groups are activated in a coordinated way.

To understand interaction techniques better, we need to understand them from the human motor control perspective. The problem it needs to solve is this: an interaction technique is a computer program that can do *anything* to the input signal. For the motor system, it is an opaque box. How can the motor system know *which* action to take? The *theory of internal models* in motor control can explain this [399]. Controlling an interaction technique requires *prediction*. That is, to pick a feedforward signal, the brain needs to predict what *might* happen. Its learning can be thought as a form of *function learning*.

By interacting with the technique, the motor control system learns – implicitly and without us being consciously aware of it – a function that maps feedforward signals and the perceived state of the controlled system. The learned model allows predicting the consequences of actions. Alas, the predictions are never perfect. While *prediction error* tends to decrease with practice, it is never zero. But at some point a sufficient level has been reached. The more accurate the prediction is, the more effective its feedforward, and the fewer iterations are needed to get to the goal. This model explains that we need lots of repetition to take control of a new interaction technique.



(a) Control without an internal model would lead to non-human-like, jerky control.



(b) Control with an internal model

Figure 26.10.: The top pane shows a simple (but wrong!) control model for interaction techniques that shows the basic components involved from a control-theoretical perspective. Bottom: To control an interaction technique, users must be able to predict the perceived consequences of actions.

26.6.2. Learning

A second hurdle concerning interaction techniques concerns human learning. In a nutshell, users are already familiar with some technique and must now invest time to learn another one. Figure 26.11 illustrates this. It shows two learning curves: initial and after change. In both, performance develops according to the power law of learning.

When a new technique is introduced, an immediate cost ensues. This is called 'performance dip'. The concept is important: Changing from one technique to use another one introduces a learning cost, which users may be unwilling to endure despite long-term benefits. From a user point-of-view, this is rational: longer-term benefits occur only much later and are uncertain. Few users can estimate what the future gains are, but everybody can experience the performance dip. 'Starting from scratch' will hamper performance right now but there is no guarantees that there will be any longer term benefit.

The performance dip has a non-obvious implication to design. It implies that users' initial perceptions of their own performance with an interaction technique tend to be negative. Hence, one should focus not only on ultimate performance, but on incentivizing users to start and persist learning the technique. This could be achieved, for example, by communicating clearly what is achievable, setting performance goals, and providing them feedback about their progress. The other concern in design is to minimize the immediate costs of the dip. By doing that, the longer-term benefits of the technique also become more pronounced.

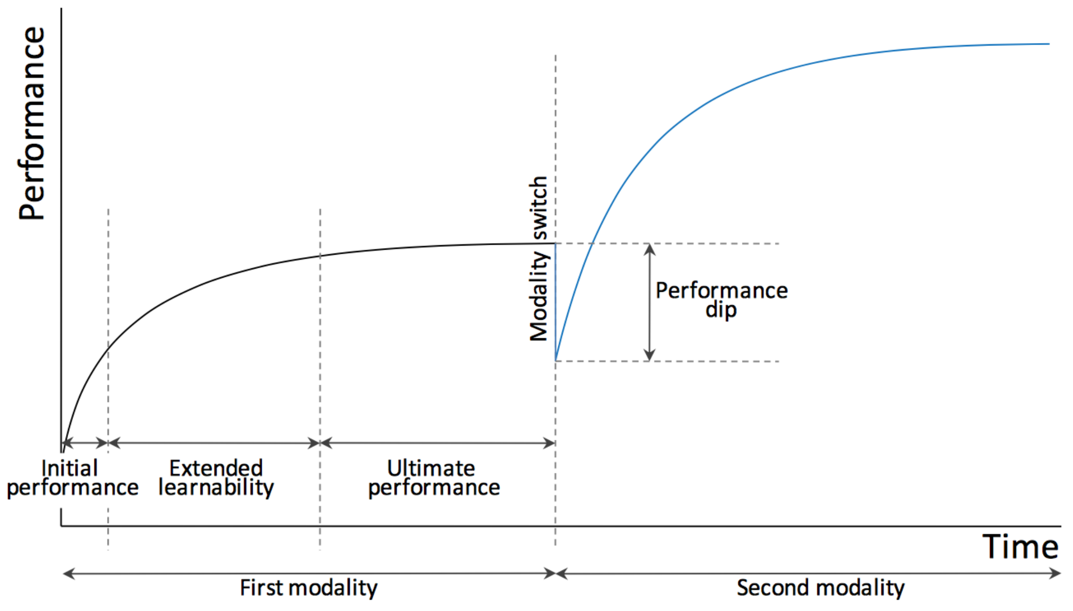


Figure 26.11.: Performance dip: Shifting to an alternative interaction causes an initial dip in performance [715].

Summary

- An interaction technique is a computer program that couples input and output processing, with the purpose of improving or enabling elementary interactive tasks.
- Many interaction techniques are about striking a desirable tradeoff between learnability and performance. Consider hotkeys versus menus, for example.
- Changing from one technique to use another one introduces a learning cost, which users may be unwilling to endure despite long-term benefits.
- Users do not need to learn how the algorithm of the interaction technique works. Instead, they learn through experience to predict the consequences of their actions (internal models theory). This takes time. However, by an appropriate feedback design, the learning process can be facilitated.
- Design should focus not only on ultimate performance but on incentivizing users to start and persist learning the technique
- Oft-researched interaction techniques include the pie menu, the marking menu, speed-dependent semantic zomoing, the CD gain function, and hypertext.
- Interaction techniques must be evaluated with realistic tasks and by relevant user groups to estimate their achievable performance.

Exercises

1. Understanding an interaction technique. The scrollbar is apparently easy to use but technically a complicated user interface element. Try to identify five different types of scrollbars. Discuss them as interaction techniques: 1) How do they couple input and output? 2) What are their relative tradeoffs in usability, discoverability, and other aspects of their value to users?
2. Trade-offs. Take two interaction techniques designed for the same purpose. Compare their relative pros and cons in terms of the design objectives given in Table 26.1.
3. 2D vs. 3D interaction. Which ideas that you know from 2D interaction could be used for 3D interaction?
4. Control order. Control order refers to the translation of input motion into camera or cursor motion: zeroth order control refers to position control, first order to velocity control, second order to velocity control, and third order to acceleration control. Your task is to implement and compare these for a panner. Which one is the best? How do you set hte parameters of each control order? Code for the panner is given on the textbook's homepage.

5. Crossing. Why is crossing-based selection beneficial? Discuss from the motor control point-of-view.

6. Buttons. Clicking may be implemented in different ways. How is it done on the device you use. How may it be useful or less useful for particular users?

7. Marking menus vs. pie menus. (This assignment requires familiarity with motor control concepts (Chapter 4).) A drawing application runs on a tablet operated with a pen. The design team wants to implement a graphical menu structure that allows users access to all commands in the software. (a) Explain the difference between closed-loop and open-loop control, including the advantages and disadvantages for each form of interaction. (b) Describe the difference between a pie menu and a marking menu. (c) Explain when interaction with a marking menu is closed-loop and open-loop. (d) The design team is considering a choice between a hierarchical linear pull-down menu and a marking menu. Describe the primary advantages and disadvantages in using either approach. (e) There are two ways to implement the hierarchical linear pull-down menu. In the first implementation, as soon as the pen tip leaves a submenu, the submenu disappears. In the other implementation the submenu only disappears if the pen tip enters the previous menu. Explain whether these implementations rely on crossing or steering actions.

27. Commands and Navigation

The crafting of text requires the writer to find the appropriate words and bring them into an organization that conveys the intended story. Sometimes writers further make efforts to organize their text in a way that readers may find their way around the text, for instance, to get an answer to a specific question or develop an overview of the text. User interfaces have related concerns. We need to name objects and actions in the user interface: A wrong word can cause a user to pick a wrong option, a poor organization of information can lead users to navigate to the wrong place. More than 50 years ago, the pioneering researcher in human factors Chapanis [144] made this connection.

> The aim of this paper is to call to attention a very large and important area of human factors engineering that is almost entirely neglected. This area consists of the language and the words that are attached to the tools, machines, systems, and operations with which human factors engineers are concerned.

Thus, the words we use in menus, virtual reality, and smartphones matter. How we structure those words into commands and into information structures that users engage with also continues to matter. What might this look like? First, numerous papers have shown the advantage of broad rather than deep web page hierarchies. Jacko and Salvendy [372], for instance, showed that a deep menu structure (that is, a menu with few items at each level but with many levels) increased, and users found the menu more complex to navigate. Second, it has been shown that providing a preview of text on a webpage—often called a snippet—improves web search. Teevan et al. [797] further showed that making visual snippets worked better, in particular for tasks where images were a prominent carrier of meaning. Third, Payne and Green [628] showed that consistency in the structure of command-language arguments helps users learn.

More generally, this chapter introduces two styles of user interface. Both depart from the observation that when a user tries to find information in a user interface, they typical have an intended action or object in mind. In Norman's seven-stage model, this is called their goal (see Section 18); in work on information search it is called an information need, that is, an unexpressed but real need for information that will help solve some problem [796]. A user may want to locate the option to bold a word or find out how to make pasta. In such cases, the user has something in mind which may only partially match what is seen in an information system.

In some user interfaces, users issue *commands* to the computer system. There are many ways to issue such commands, such as using a command-line interface or navigating to a desired command using a menu structure. Search, such as on the World Wide Web, may also be understood as a form of command-driven user interface. To help deal with command-based systems, this chapter introduces the cognitive dimensions of notation framework as a design vocabulary that provides terminology that allows designers to reason about the notation of interactive systems, including, but not limited to, user interfaces that employ commands. The central concern about commands is *appropriate naming of actions and objects.*

The second style of user interface is *navigation.* In it, users explore the user interfaces to discover commands and information. As such, navigation support is often critical to a well-functioning user interface system. This chapter will discuss common navigational techniques to help users discover and issue commands and navigate hypertext. The central concern for navigation is *appropriate organization of information.* Appropriate here means that it is recognizable to people and efficient to explore.

27.1. Naming Objects and Actions

Why is naming things in user interfaces challenging? The paper box explains the vocabulary problem, one difficulty in naming objects and actions in user interfaces, together with the issues of polysemy (the same word means different things), the role of context in interpreting words, and many other issues. Therefore, it is difficult to find clear and precise descriptors for user interfaces.

Paper Example 27.1.1 : The variability in naming objects and actions as a central user interface challenge

Imagine that you were to name a program or application that would tell you about interesting things to do. What would you choose if you could only use 10 characters or less? If you were to pick alternative words for "love" or "motorcycle", what would they be? Try to write down your answers before continuing.

Furnas et al. [261] considered these questions important to HCI. They argued that the way we name the objects and actions that we are interested in engaging with in user interfaces is important to many user interfaces. But when they investigated how people named objects and actions (as you just did in the exercises above), they discovered something surprising. If you ask two people to spontaneously name commands, as you just did, they will agree with less than .2 probability. Thus, only every fifth command may overlap with that of another person. They called this "the vocabulary problem". The issue is that finding a command that will work well for many users is very difficult.

Furnas et al. [261] went on to consider a solution that allowed users to create unlimited aliases for particular commands. In many ways, that is similar to the way search works on many current devices. However, the general insight is that picking the appropriate command names is extremely difficult.

Some general ideas for thinking about naming objects and actions may be given. First, we may think about naming in terms of information scent (see **??**). Recall that information scent concerns how well a word or phrase creates associations (i.e., a scent) of what they subsume (e.g., what command they execute or what menu items they may be expanded into).

Another general principle to think about for naming is articulatory distance. This concept quantifies the distance between the physical form of the name or command and the command that the user wants to express. Thus, we want the intended command to be as similar as possible to the user's intention.

Another insight into names is that specific names are preferable over more general names. Black and Moran [74] showed that command names that are infrequent and discriminative are more likely to be recalled than more frequent words. Thus, we should use "insert" rather than "add" as a command for placing text into a document. Recall, of course, is crucial for users who use an interface intermittently.

Furthermore, it is better if names for objects or actions are meaningful rather than purely designating a command by convention or through some symbol. Black and Moran [74] showed that non-words worked relatively well when only a few commands needed to be learned, but that it had high error rates. Also, comparisons of the performance of keywords (e.g., "CHANGE 'KO' TO 'OK' ") to symbols (e.g., "RS: |KO|, |OK|") usually favor the keyword [455].

27.1.1. Techniques for Finding Names

Given all this complexity, how do we find appropriate names for labels in a user interface, for commands to a spoken-language system, or the right names for a menu item?

The first approach is to pick the names that users employ in real life for objects and actions. Such names may be found during user research (see Part 3). This is a straightforward idea, captured in heuristics such as “speak the users' language” [576].

Another approach that is often used in HCI is *elicitation study* [886]; it is also useful to figure out the command names. The basic idea in elication studies is to elict from users how they will execute a particular command and study the agreement across users on those commands to decide on a distinct set of commands that users will produce spontaneously with the highest likelihood. For instance, we may show users a command we would like them to execute (e.g., "make a copy of this file") and ask how they would name that action (e.g., "duplication"). We then use the majority of names for the command. Elicitation studies do not address the vocabulary problem; rather, they try to find a solution.

Paper Example 27.1.2 : Programming as a naming problem

As we shall see, the concerns of naming commands have rather direct implications for interfaces that are driven by commands. However, similar concerns apply to many other areas, including programming languages. Myers and Stylos [557] looked specifically at application programming interfaces (APIs) and considered them a form of user interface.

The key idea is that APIs serve as an intermediary between the user (in this case, the programmer) and an interactive system (in this case, whatever the API provides access to). The design of the API, therefore, can be considered from the viewpoint

As one example, the naming of programming constructs should make it easy to relate them to the programmer's world. Myers and Stylos [557] gave an interesting example of this.

> For example, the most generic and well-known name should be used for the class programmers are supposed to actually use, but this is violated by Java in many places. There is a class in Java called File, but it fis a high-level abstract class to represent file system paths, and API users must use a completely different class (such as FileOutputStream) for reading and writing.

As another example, Myers and Stylos [557] discusses consistency for the following two methods; it is not relevant if you know the particular programming language.

```
void writeStartElement(String namespaceURI, String localName)
void writeStartElement(String prefix, String localName, String namespaceURI)
```

These two methods use a different ordering of elements, violating the principle of consistency and making the transfer of learning from one method to the other difficult. In short, it violates what we know about learning and about consistency in user interfaces. Moreover, in this specific case, it was even hard to recognize that an error was made because both arguments are of the type cmttString.

27.2. Command-Line Interfaces

A command-line interface (CLI) is a text-based user interface that enables the user to interact with the computer system by typing commands. The history of the command line interface begins with the invention of the telegraph. Telegraph operators were tasked with transcribing Morse code into text.

A command-line interface can be decomposed into a prompt, a command, and n parameters:

$$\texttt{prompt command parameter}^1\ \texttt{parameter}^2 \ldots \texttt{parameter}^n$$

Prompt The *prompt* is a system-generated indication that the system is awaiting a command from the user. Prompts can be minimal, providing only some indication (such as a beep) that the system is awaiting a response, or they can be elaborate and provide additional information to the user, such as the some aspect of system states.

Command The *command* is the instruction provided by the user to the system.

Parameter list The *parameter list* (sometimes called an *argument list*) is a set of command-specific additional instructions provided by the user, which will affect the execution of the command.

A simple example of a command line is the interpreter in early adventure games, such as the early graphical adventure game *King's Quest 1*, designed in 1984 by Roberta Williams. In this game, the player controls a character representing a prince that can walk around in a graphical scene. The player can interact with the environment, objects in the environment, and non-playable characters in the environment using a command-line interface. For example, if the player has moved the prince to the vicinity of a graphical depiction of a candle, the player can pick it up using the following command:

```
>take candle
```

Here `>` is the prompt, `take` is the command, and `candle` is the parameter. If the candle cannot be picked up, the system responds with a message. If the prince was not close to a candle or for some reason would be unable to pick it up the system would respond with an alternative message.

This user interface is an example of a basic command-line interface. The prompt is merely a minimal indication that the user is expected to type commands, and it does not change depending on context, nor does it ever indicate any system state. The list of commands, *vocabulary*, is limited a small fixed set of commands, such as `take`, `look` and `talk`. These commands may be executed without a parameter. For example, the user may type `look`, which without any parameter provides an overall description of the graphical scene depicted on the screen. Alternatively, the user may provide an argument, such as an object, for example, `look candle`, which provides a description of a candle, assuming such an object is present in the graphical scene.

Command-line interfaces processes commands using an *interpreter*, a software component that receives text input from the user and outputs the command and the parameters the user typed. This interpretation is frequently referred to as *parsing* and, therefore, the interpreter is sometimes also referred to as a *parser*. If the interpreter cannot make sense of the user's input the system outputs an error message. Historically, these error messages were often obtuse, such as the ubiquitous "syntax error" message, which merely indicates to the user that the system failed to parse the response text. However, error messages can be more detailed and helpful, such as "`Command 'save' expects a filename`" (if the user forgot to indicate a filename) or `There is no command 'sve'` (if the user misspelled the command).

In the case of *King's Quest 1*, the interpreter knows a specific set of commands (which are all verbs) and in addition knows which commands accept which arguments and when.

If the user type a command that is not present in the interpreter's command list, a generic response such as "`I do not understand that`" or "`I cannot do that`" can be easily generated by the system. In this case, such exploration and experimentation with commands and their parameters is part of the game design.

However, even such a simple interpreter has a set of design issues that need to be resolved, such as:

Availability How does the user know which commands are available?

Naming How does the user know the names of the commands the system understands? For example, the interpreter may understand `take` but not `pick up`.

Learning How does the user learn the availability of the commands and their parameters?

Recall How does the system support the user in recalling commands and their parameters?

Syntax How does a user know which commands expect parameters and the form of these parameters?

Transparency How does the system convey to the user the way in which it interprets commands, making it possible for the user to both understand what is achievable within the command-line interface and be able to diagnose why a particular command input does not result in an intended response.

As the command-line interface becomes more complex, these design issues become more important. For example, a common example of a command-line interface is a *command-line shell*: a software that allows users to interact with the operating system and applications by typing commands. Such shells have not only many commands but also elaborate syntax, which may be difficult for users to comprehend and even more difficult to understand when the shell fails to interpret the user's commands.

Further, such shells may have several categories of commands, such as:

Built-in shell-commands A shell typically provides a set of commands that are integral to the shell itself. An example of such a command (in the shell *Bash* for Linux) is `echo`, which outputs a provided parameter to the shell. For example, "`echo hello`" results in the shell outputting "hello".

Operating system provided tools In addition to built-in shell commands, most operating system provides a range of services for connecting and disconnecting devices, managing the file system and other services, and typically an operating system provides command-line applications that allow a user to interact with such services. These operating system-specific tools are always present in a working installation. An example of such an application in Linux is `cp` that allows the user to copy a file. For example, "`cp file1 file2`" creates a copy of `file1` with the name `file2`.

External applications In addition to built-in commands and operating system provided tools, users may also have the ability to access additional functionality by interacting with external applications using the command-line. An example in Linux would be `cpp`, which is a compiler that can be instructed to compile code by providing it with command-line parameters.

When a command-line interface increases in complexity the design issues—availability, naming, learning, recall, syntax—in turn increase in significance. For example, built-in commands should use a systematic naming scheme and syntax though this is by no means guaranteed. This challenge of ensuring consistency and standardization across commands becomes more difficult when a range of additional operating system provided tools are provided, some of which may be legacy tools with previously established conventions or third-party provided tools. External applications exacerbate the situation further since the naming conventions and syntax of external applications cannot be directly controlled by the command-line interface and is therefore likely to vary despite any availability of indirect control, such as user interface design guidelines.

27.3. Organizing Information

How to organize information has been a topic for more than 100 years, starting with considerations on how to structure increasing amounts of information in books and other publications. With the invention of the internet, those considerations have spread to the digital world, too. They are often called information design or information architecture [691]. We will just call this concern *organizing information* and offer a high-level view of some considerations and tools.

In general, the challenge of organizing information assumes that we have items of information (e.g., commands, menu items) that we want to organize in some way (e.g., in a list, in a hierarchy).

27.3.1. Ordering Information

The simple case is where we have a set of items that we need to order. If you look at the items in the first menu you can find on your computer, they will have an order. How do we decide that order so that it make the user interface the best possible?

The most general guideline is to *group* items that belong together. Such a grouping may be functionally related items (e.g., "copy", "cut, "paste"). In most empirical comparisons of ordering, grouping is the most effective ordering principle.

Another approach is to use *importance* of the items. One might infer importance from the *frequency* with which users select options. However tempting it seems to change the order of items, for instance based on selection frequency, Somberg [763] described an early study that showed this strategy performed poorly. The study compared finding item in one of four menu orderings: alphabetic, frequency of selection, random, and positionally constant. Surprisingly, positionally constant placement worked the best, in particular as users practiced more finding the item.

While *alphabetical* ordering is easy, it is suitable only for a few tasks. For most other tasks, there is either some way of ordering items that better match users' typical tasks or the information can be placed into hierarchies—if you have tried searching for a country in an alphabetical list of countries, you know that hierarchies can be useful.

27.3.2. Placing information in hierarchies

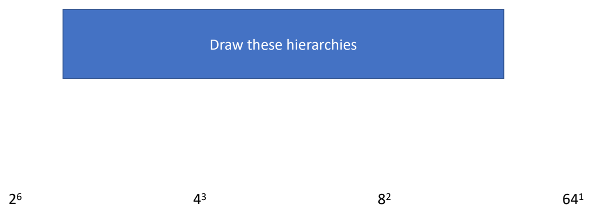
Oftentimes, we have so many items to organize that placing them into a hierarchy is unavoidable. As for organization within a list, items that are semantically or functionally associated should be placed in the same panel or submenu. Place the most informative items of a group to the top of a panel. Apply this principle to every level of a hierarchy.

Another important idea is that hierarchies should give labels or hints that allow users to quickly and without error decide whether the target they are looking for belongs to the tab or not. In foraging terms, design the 'information scent' of the entry points of a panel. For example, when looking for 'Save', if the menu label says 'Window', scent is low and the user can quickly dismiss the whole panel. If it instead says 'General', scent is higher and a user might occasionally look into the panel, in vain.

One important decision in placing information into hierarchies, be it a menu or a web-site structure, is whether the hierarchy is broad or deep. As suggested by the paper example below, and many other studies, broad menus appear to be preferable for most situations. Why is that so? A broad menu structure makes it easier to compare alternatives visually, rather than to go down a path in the hierarchy only to discover that an alternative would be better. A broad menu also offers fewer choice points for the user. Finally, the labels for the parts of the hierarchy are perhaps easier to make high-scent because they are fewer and therefore somewhat broader.

Paper Example 27.3.1 : Breadth/depth tradeoffs in menus

Miller [532] presented one of the first studies of the tradeoffs between organizing hierarchies to be deep (few choices per level) or to be broad (many choices per level). Miller had 24 participants try one of four menus, each containing 64 items. The menu hierarchies varied between two choices at six levels, four choices at three levels, eight choices at two levels, and 64 choices at one level (see below). The menu items were common English words



Miller [532] had 24 participants use one of the four menu organizations, where they had to select the 64 goals one at a time for a total of four times (so-called blocks). The purpose of these blocks were to see if participants improved their ability to find items quickly and without errors as they gained experience with the hierarchy.

The results of the study indicated an U-shaped relation between increasing depth of the organization and both task completion times and errors. The hierarchies with four or eight choices were thus optimal, leading to the conclusion that a tradeoff between choices and organization exists. In particular, seeing only leaves of the hierarchy (64 choices) is ineffective, as is a series of many choices.

27.3.3. Hypertext

Text which allow the user to retrieve further information when interacted with is known as *hypertext* [569]. Hypertexts are connected using *hyperlinks* (often simply called “links”), which are particular elements that trigger further retrieval. Hyperlinks consisting of text are usually denoted in some unique way to provide sufficient perceived affordance so that the user can realize a hyperlink is present.

Hypertext is ubiquitous on the world wide web (WWW) but not limited to such use. The traditional hyperlink on the WWW is blue and underlined to indicate its unique function in comparison to regular text, which may be selected or even edited, but will not trigger retrieval of further information if interacted with. Hyperlinks may consist of text, images, and videos. The broader term *hypermedia* is sometimes used to denote more general usage.

Since hypertext documents can link to other hypertext document, and such hypertext documents, can in turn link back to the original source hypertext documents, the navigational structure of hypertext document collections can be complex. Mathematically, hypertext document collections form directed graphs. While a simple hypertext document collection may have a relatively straightforward tree structure, in practice, hypertext documents quickly form complicated directed graphs that may be difficult for users to navigate without aid.

While originally a highly active area of research, today hyperlink document navigation has been largely standardized using a set of well-known navigational structures, such as the use of a navigation bar with *forward* and *backward* buttons and an address bar. A similar structure is also used in many file browsers.

[66] provides an overview of common navigation techniques for hypertext. The first technique is the use of presentation, layout, and links to convey information, orientation cues and choices. The second technique is the use of maps and indexes to provide an overview of the hypertext document collection. The third technique is the use of bookmarks to allow the user to quickly retrieve a particular hypertext document. This technique allows the user to mark the parts of the hypertext document collection that are of particular interest to the user. The fourth technique is the use of thumb tabs, which allow the content creator to mark the parts of the hypertext document collection that may be of interest to users in general. This can be achieved by, for example, including hyperlinks at the side, top or bottom of the hypertext document that links to key information. The fifth technique is the use of margin notes that allow the user to annotate hypertext documents. The sixth technique is use of breadcrumbs, a navigational aid that reminds the user of previous locations the user has visited in the hypertext document collection. An initial “crumb” will often indicate an overall summary page or entry page while subsequent crumbs reflect the user’s navigation deeper into the structure, for example “Animals”, “Mammals” and ‘Cats‘.

27.3.4. Techniques for organizing information

One of the main techniques for discovering organizations of information that match users’ organization is called *card sorting.* The basic idea in card sorting is to provide users with descriptions of the information to be organized, each piece of information represented separately (e.g., on a card). Users are then asked to sort those into groups of similar items, possibly naming each group. Across users, these grouping can give a sense of how consistently users see similarities of pieces of information and what groups of information should be called. If users are consistent, the groups may be used to organize information in a user interface.

Typically, the card sort process goes as follows [894, 616].

1. Put pieces of information on cards; the cards may be digital or physical.

2. Find users that can help group the cards; any number will do, although 20 to 30

persons are often recommended [812]. It is important that the users understand the cards.

3. Ask users to group the cards. They may be asked to "“Please group these items as you would expect to find them on our corporate intranet" or "group things together that seem to be similar in some way".

4. Ask users to name groups; those names may be used in the user interface and may be helpful to understand why cards are grouped together.

After this, various statistical methods are available for calculating which groupings that seems best. If we have x cards with information pieces and have asked y users to group them, we obtain a matrix of $x \times x$ where each entry is the number of times (between 0 and y) two items were grouped. This matrix can be used for cluster analysis, for instance to derive a hierarchy of pieces of information or to find a set of k groups through k-means clustering. For more information, see for instance Paea et al. [616].

It is also possible to validate the card sorting results by having users place pieces of information into existing categories (so-called closed card sort). This may be considered a check of the reliability of the grouping (see Chapter 10).

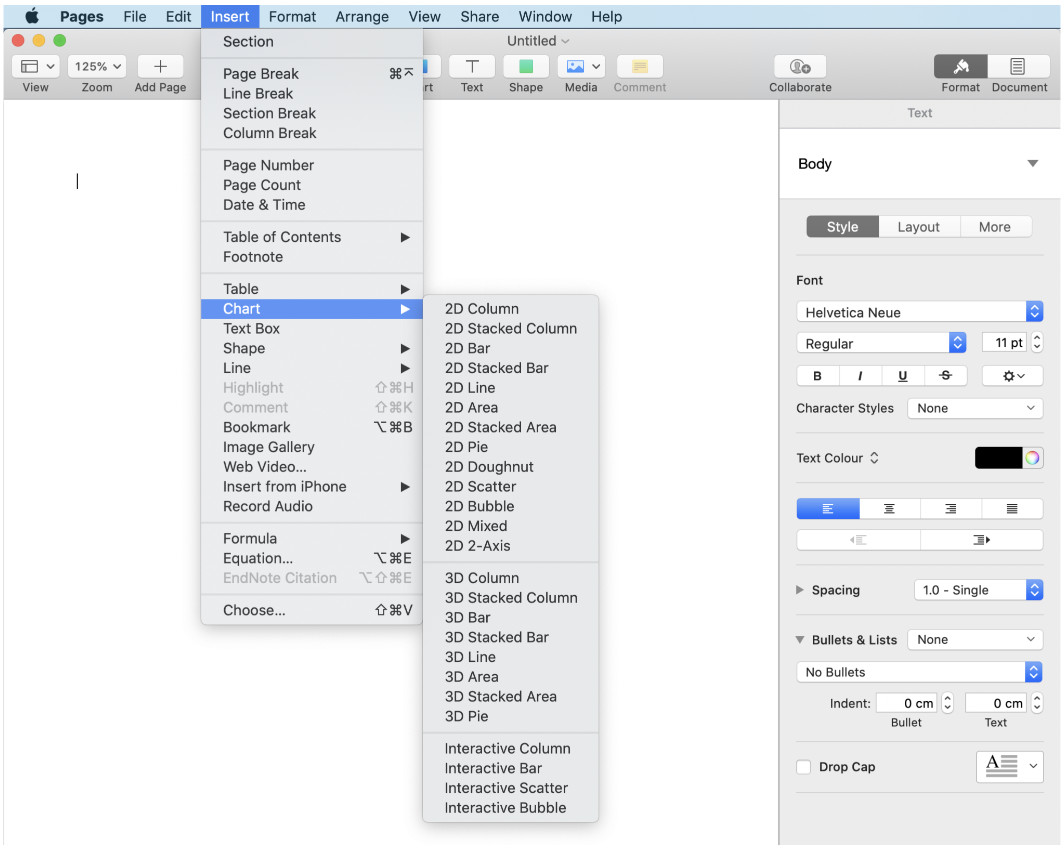
27.4. Menu User Interfaces

Command-line interfaces (CLIs) can be efficient, assuming that the user can take the time to learn them. As a result, they are useful specifically in professional applications where skill development or training can be expected, such as system administration [48]. However, for many other cases, learnability can pose a barrier. It is time-consuming to learn command names and effortful to recall them. The principle of direct manipulation (Chapter 28) would be both learnable and less effortful. However, it does not offer as much room for improving performance. As a middle ground, interactive menu systems have evolved as a mainstream technique for command invocation in graphical user interfaces.

We are all familiar with menus. In general, *a menu* is a set of selectable options, like a menu of desserts in a restaurant. *An interactive menu* displays its options on a computer display for selection with a pointing device [36]. Interactive menus first appeared in 1968 in AMBIT/G, a visual programming language, but they were popularized much later by the Xerox Star in 1981 and later the Macintosh computer in 1984 [36].

As a common type, Figure 27.1 shows an example of a hierarchical linear pulldown menu. It shows several *menu commands* organized into *menu panels*. Within each panel, there are *groups* of commands indicated by (horizontal) *separator lines*. The menu is linear because commands in a panel are organized vertically. The menu is hierarchical because there are multiple panels. Some panels o open *submenus*. The menu is a pull-down menu because it opens in place by expanding from top to bottom.

This real-world example illustrates well the main benefit of interactive menus: they allow novice users to familiarize with options simply by opening menu panels and reading the items. Items are made available via *recognition* not *recall*, which is effortful (see



Zoom: 125%

Body

Font: Helvetica Neue, Regular, 11 pt

Character Styles: None

Text Colour

Spacing: 1.0 - Single

Bullets & Lists: None

No Bullets

Indent: 0 cm Bullet, 0 cm Text

Figure 27.1.: An example of hierarchical linear pull-down menus in *Pages*, a word processing application.

Chapter X). There is no need to memorize command names. However, recall can be used to further boost performance. *Keyboard shortcuts* (e.g., CTRL-C for copying and CTRL-V for pasting) allow users to drastically improve performance for oft-used commands. This mode, although it requires some effort to learn, is optional.

Common characteristics of interactive menus are:

A finite set of options Typically between a handful and a few hundred, but can go up to a few thousands in professional software;

Visual and textual presentation Options are presented visually on display using text, icons, and interactive feedbacks (e.g., hoverover cues);

Interactive exposition of options Menu options can be shown fully or exposed interactively as a response to the user's input;

Transience Menus are often expanded in place and dismissed after use, saving screen estate;

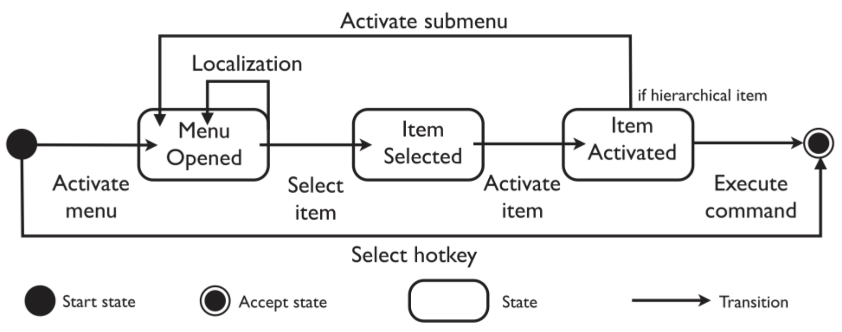


Figure 27.2.: In menu interaction, user traverses via options using a pointer.

Multimodality Alternative ways to access the menu can be offered, such as via voice commands or keyboard shortcuts.

27.4.1. Menu techniques

In the menu design shown in Figure 27.1, submenus are opened by hovering over the parent option. This is an instance of *a menu technique*, an interaction technique for menu navigation and selection. It illustrates two principles of menu systems: interactive exposition and transience. Submenus are exposed and dismissed by bringing the cursor on top of a parent menu.

To issue a command, the user has to perform a series of actions: 1) trigger the menu; 2) identify the menu item with the desired command; 3) navigate to the menu item with the desired command; and 4) select the menu item. While an expert user would know where a desired menu item is located, the user would still need to navigate to this item. This is inefficient compared to a direct invocation method, such as a keyboard shortcut.

Menu techniques can be divided into two main classes, depending on which aspect of the series they support. Techniques for navigation help the user explore the menu panels. For example, keyboard navigation can be offered, or hover-over -based exposition like the one above. Techniques for selection help the user select an item. In a basic form of interactive selection, shown in Figure 27.2, the item that is currently being pointed at is highlighted (activated), helping the user to keep track of the current selection. When the cursor is brought over an item, it can be interacted with.

The design of a menu techniques can help find a suitable tradeoff between performance and learnability. While in theory pull-down menus can cascade indefinitely, in practice pull-down menus are limited in the number of commands that can be made available to the user. Careful steering of a control cursor is required in order to navigate a hierarchical menu structure [7]. This limits performance.

27.4.2. Keyboard Shortcuts

A keyboard shortcut is a keyboard combination, such as `Ctrl+C`, that allows a user to directly invoke a command. The advantage with a keyboard short-cut is that they allow users to immediately invoke a command without having to navigate a structure. This provides efficiency to expert users.

The primary disadvantage of keyboard shortcuts is that the user has to consciously learn the keyboard shortcut. To aid this, keyboard shortcuts are often, but not always, shown next to a command as part of a menu item. This allows the user to learn about the keyboard shortcut at the time when they are interested in the command. On the other hand, at that point it is easier for user to simply select the menu item, which means learning the command only happens due to continuous user effort.

Another disadvantage with keyboard shortcuts is that users may forget them and unless they are used frequently.

27.5. Notational Systems

A *notational system* is a system with graphical elements composed according to rules, sometimes referred to as a visual grammar [76]. We introduce such systems here because both commands and navigation structures may be understood and analyzed as notational systems.

An example of a notational system is a domestic heating controller. A typical heating controller consists of a set of buttons for controlling hot water and radiator heat using either direct controls or by programming. The controller provides buttons allowing the user to turn on either the hot water, radiator, or both systems, for one hour. To program the controller to turn heating or or off at particular times during a day, the user has to move a panel, revealing a set of buttons to program the controller.

Such a heating controller can be analyzed in terms of its *notation*: how it allows the user to operate it by, for example, modifying the times hot water is heated or the radiators are turned on and off. The cognitive dimensions of notation framework [76] provides a design vocabulary for reasoning about such notational systems.

An example of how this framework can be used is posed by the following quote ([76]:

"Back in the 80's one of us (TG) was shown a system that he was sure wouldn't work. Yet the received wisdom of 80's HCI seemed to have no explanation of the problems he foresaw. It was a speech-to-text dictation system for use with the Pascal programming language. The speech-recognition technology appeared to be entirely adequate for this restricted language. What problem could there be?"

27.5.1. Types of Notation Use Activities

Incrementation Adding cards to a cardfile, formulas to a spreadsheet or statements to a program

Transcription Copying book details to an index card; converting a formula into spreadsheet or code terms

Modification Changing the index terms in a library catalogue; changing layout of a spreadsheet; modifying spreadsheet or program for a different problem

Exploratory design Sketching; design of typography, software, etc; other cases where the final product cannot be envisaged and has to be 'discovered'

Searching Hunting for a known target, such as where a function is called

Exploratory understanding Discovering structure or algorithm, or discovering the basis of classification

27.5.2. Cognitive Dimensions of Notation

A lack of terminology may make it difficult to reason precisely about a notational system's positive and negative qualities. *Cognitive dimensions of notation* form a *design vocabulary* that help designers and researchers to discuss, assess and critique notational systems. The cognitive dimensions are the following:

Viscosity *Resistance to change.* Viscosity captures the extent of actions the user is required to carry out to achieve their goals in a system. *Repetition viscosity* means that a change requires many actions of the same type, such as for example, selecting each cell in a spreadsheet individually and assigning them a color attribute. *Knock-on viscosity* means that further distinct actions are required to restore consistency following a change. The more involved and complex it is for a user to make changes in a system, the higher the viscosity.

Visibility *Ability to view components easily.* The overall visibility provided by the system is determined by the saliency of visual elements in the interface and the ability to reveal visual elements easily. A system that makes it difficult to view elements of the system has low visibility.

Premature commitment *Constraints on the order of doing things.* The degree of flexible a user has in carrying out operations in any preferred order. A system demanding many operations to be carried out in a very specific order exhibits a high degree of premature commitment.

Hidden dependencies *Important links between entities are not visible.* If certain elements in the system rely on other elements, such dependencies should ideally be evident to the user. A system that has many hidden dependencies is not making it clear to the user that many elements in the system are reliant on other elements or factors present in the system.

Role-expressiveness *The purpose of an entity is readily inferred.* Role-expressiveness describes the ability of a user to inspect an element in the interface and understand its purpose with little or no guidance. Role-expressiveness is closely related to, but not identical to, perceived affordance. While perceived affordance relates to the immediate uses of a visual element, role-expressiveness may be provided by the system by other elements means than the element itself, for example, by providing a description in the vicinity of an element. The easier it is for a user to understand the purpose of the elements in the system the higher the role-expressiveness.

Error-proneness *The notation invites mistakes and the system gives little protection.* The error-proness of the system can manifest itself in four different ways. First, the system can be fragile and easily reach an erroneous state. Second, the system can make it easy for a user to commit an error. Third, it may be difficult for the user to realize there is an error. Fourth, there may be either no provision for rectifying an error or only a cumbersome error correction facility. These four error-proneness categories can interact. For example, a system may make it easy for a user to commit an error and make it difficult for a user to realize that an error has occurred. The higher the extent these four categories are present in a system the higher the error-proneness.

Abstraction *Types and availability of abstraction mechanisms.* Abstractions are definitions of underlying notations. For example, a style sheet in word processing allows a user to create a new style, such as a "body text" style that encapsulates text attributes: a specific font, font size and font style for the running text in a document. Instead of the user formatting the text by manually assigning these individual text attributes, the user can assign the text a style. As this example shows, abstraction may necessitate additional user interface components, in this case a style sheet manager that allows the user define and redefine styles. The more intricate the types of abstraction are in the system and the more abstraction mechanisms that are made available to the user, the higher the abstraction provided by the system.

Secondary notation *Extra information in means other than formal syntax.* A system can provide mechanisms for users to record additional information that is unanticipated by the notation designer. For example, program code allows the insertion of comments that are uninterpreted by the system. Such comments can then be used to record a variety of information, such as an explanation of the purpose of a variable or function. The more advanced mechanisms in place for the user to record additional information, the higher the secondary notation support provided by the system.

Closeness of mapping *Closeness of representation to domain.* The closeness of mapping describes how well the notation maps to the phenomena that the system is attempting to capture. The better the match between the notation and these phenomena, the higher the closeness of mapping.

Consistency *Similar semantics are expressed in similar syntactic forms.* It is easier for the user to infer the correct application of a notation for a task if the notation of a similar task is of a similar form. For example, if similar commands are named and presented in a similar manner it is easier for the user to find them. The higher the consistency of a system the more the system tends to express similar notation in similar manner.

Diffuseness *Verbosity of language.* Notations can be overly redundant, resulting in overly long expressions in a text-based notation or disproportionate use of screen real estate. The more redundant the notation, the higher the diffuseness.

Hard mental operations *High demand on cognitive resources.* The more difficult it is for the user to conceptualize and understand tasks or components in the system, or the system as a whole, the higher the demand on the user's cognitive resources. For example, deeply nested structures, complex interrelationships between elements and obfuscated presentation all contribute to higher working memory demand. The more such demands are made by the system the more the system provides hard mental operations.

Provisionality *Degree of commitment to actions or marks.* The ability of the system to allow the user to temporarily explore different solutions or designs is measured by its provitionality. The higher the provisionality, the more the system supports the user providing noncommittal input for the purpose of exploration or sketching.

Progressive evaluation *Work-to-date can be checked at any time.* In a non-trivial system it is beneficial for the user to receive incremental feedback on progress. For example, when writing program code in an integrated development environment, an online syntax checker can provide early feedback to the user if the code expressions have syntax errors. The more a system supports providing such incremental progress feedback, the higher its progressive evaluation support.

The easiest method to understand how cognitive dimensions of notations can be applied in practice, is to use them to analyze and contrast how two different notational systems support the user in achieving a set of tasks.

27.6. Recognition versus Recall

One of the big tradeoffs that has run through this chapter is that between recognition and recall. When users *recognize*, they identify an item of relevance based on some cue for what they want to do among a set of items. This is what we all do when we use menus. When users *recall*, they come up spontaneously with names for commands or descriptions of objects. This is what we all do when we search the World Wide Web. But from an HCI perspective, the difference between recognition and recall in user interfaces represents a fundamental tradeoff.

The benefit of recognition follows from what we understand about human cognition (see Chapter 5). For most people, recognizing "Ljubljana" as the capital of Slovenia is easier than recalling the word and its spelling. Sometimes, recognition is referred to as knowledge-in-the-world; rather than relying on memory, we rely on representations of knowledge in our external environment.

The benefit of recall is that it scales well. Independently of the number of items, recall works. The items to be recalled does not take up space on a display. Recall typically requires practice: We learn capitals only by hearing them repeated. Recall is sometimes also called knowledge-in-the-head; we rely on memory rather than external environment.

The dynamics of recall and recognition is surprisingly complex. We all know situations where we rely on knowledge in the world rather than knowledge in the head; looking up facts on your phone rather than thinking about them is one example. A study by Sparrow et al. [766] explored this effect. Participants were asked to learn trivia and type them into a computer. Half were told that the computer would remember what they typed, the other half that what they typed would be erased. The participants who believed that the information had been saved performed *worse* on a task to recall the trivia. Thus, representing information externally affects how well we remember it. Gray and Fu [282] showed a fascinating effect that appears opposite. People were willing to rely on imperfect memory in interacting with a video recording machine when the cost of accessing knowledge in the world increased. Thus, we may prefer to recall, even if recalling information leads to poorer performance.

Because of these differences in display space requirements, practice, and rational tradeoffs in what we remember, the suitability of recall versus recognition as a strategy for a particular user interface is impossible to answer in the general. Therefore, we cannot, in general, say that user interfaces that rely on commands (like command-line interfaces) or on navigation (like menu interfaces) are better or worse. It depends on the users, their activities, and the context in which they act.

Summary

- In user interfaces, naming the objects and actions is an important activity, in particular because users are likely to call objects and actions different things, the so-called vocabulary problem.

- Recall of command names is facilitated by naming commands using familiar terminology and placing them in well-thought hierarchies.

Exercises

1. Recall vs. recognition. Come up with three cases where recall is needed in HCI. Then think how, by user interface design, you could transform this to recognition.

2. Command names. Consider the following actions and imagine that they would be triggered by a single command action:

   a) Copying a photo from a mobile phone into a laptop computer.

   b) Sorting all files in a directory in reverse alphabetical order.

   c) Selecting all files in a directory that are image files.

   Find one other person and work independently to propose a single command name for each action. Compare your proposed command names. How many command names are the same? Do you agree among yourselves that all proposed command names are reasonable?

3. Cognitive demands of notation systems. Choose six of the cognitive dimensions of notations and articulate some of the user interaction issues that contrast command-line interfaces and a graphical user interface.

4. Notational system design. Consider the following two notational systems:
   a) A user writing program code in an Integrated Development Environment (IDE).
   b) A user working with formulas in a spreadsheet.

   The task is to implement mathematical expressions where the arguments may be scalar or vectors and the functions used are a subset of commonly used mathematical operators: addition, subtraction, multiplication and division, exponential, square root, absolute value, and standard deviation. Use all the cognitive dimensions of notation to analyze how these two different programming environments can support the programmer in carrying out this task.

5. Cognitive dimensions. At the end of the section on notational systems there is a quote from Blackwell and Green [76]. The authors recall an anecdote about a system using speech-to-text to support programming in the programming language Pascal (the precise programming language is unimportant). Even though the speech-to-text functionality itself "seemed adequate" the authors appear certain the particular system would have HCI problems. Use cognitive dimensions to articulate some of the problems you could foresee with such a system.

6. Menu systems. Articulate the positive and negative qualities of hierarchical linear pull-down menus compared to the design of a marking menu. Are there particular use contexts where one design would preferable instead of the other one? What would those use contexts be and what factors in those use contexts would result in one interaction technique being preferable in favor of the other?

7. Command discovery. A major problem with command-lines is supporting users in discovering which commands are available and their functions. Can you think of design solutions that could mitigate this problem? Can you articulate why your design solutions would improve user experience using, for example, the cognitive dimensions of notation as a design vocabulary?

28. Graphical User Interfaces

We are all familiar with *graphical user interfaces* (GUIs). GUIs are the type of user interface that dominates workstations, laptops, and touchscreen devices, including mobile phones and tablets. GUIs allow users to efficiently carry out many everyday computing tasks with ease, such as copying and pasting information, starting applications by selecting them, working in multiple windows at the same time, and using word processing and spreadsheets with easy access to hundreds of functions.

In addition to being a part of our daily lives, graphical user interfaces have been the subject of much research from the 1980s and until now. Highlights of this research include the following.

- GUIs pioneered the idea that we can refer to objects by their location, rather than by their name. In some circumstances, this is a powerful way to learn and use user interfaces.
- The benefits of metaphors in graphical user interfaces have been much researched; examples of such metaphors are the desktop (for organizing files) and the thrash can (for deleting files).
- Some studies have shown that GUIs are more useful than command-line interfaces. Rauterberg [674] compared a GUI with an interface using menus and function keys; participants used between 18% and 88% less time to complete tasks with the GUI.

GUIs evolved from command-line interfaces (covered in Chapter 27). Although command-line interfaces (CLIs) are efficient for experts, they have several limitations. Users have to learn the command set, or part of it, including their arguments, and their syntax, in order to instruct the computer system. They are impractical for many object-manipulation techniques, such as selecting a region of an image, selecting text in a document, or cells in a spreadsheet. They make it hard to work on multiple tasks simultaneously, since there is no easy way of switching between tasks. Moreover, commands tend to be difficult to remember and are error prone.

Computing devices started as relatively simple calculation devices, and their input and output systems were transplanted from teleprinter devices used to relay text messages. However, as processing power and memory capacity increased, it became possible for computers to perform more sophisticated tasks. In parallel, new display technology and video processing enabled graphical display, while new input technology, such as lightpens and computer mice, provided users with a way to make precise selections on graphical displays. However, to allow users to achieve their goals effectively, efficiently, and safely, it was recognized that there was a need to reimagine the way users interact with computer

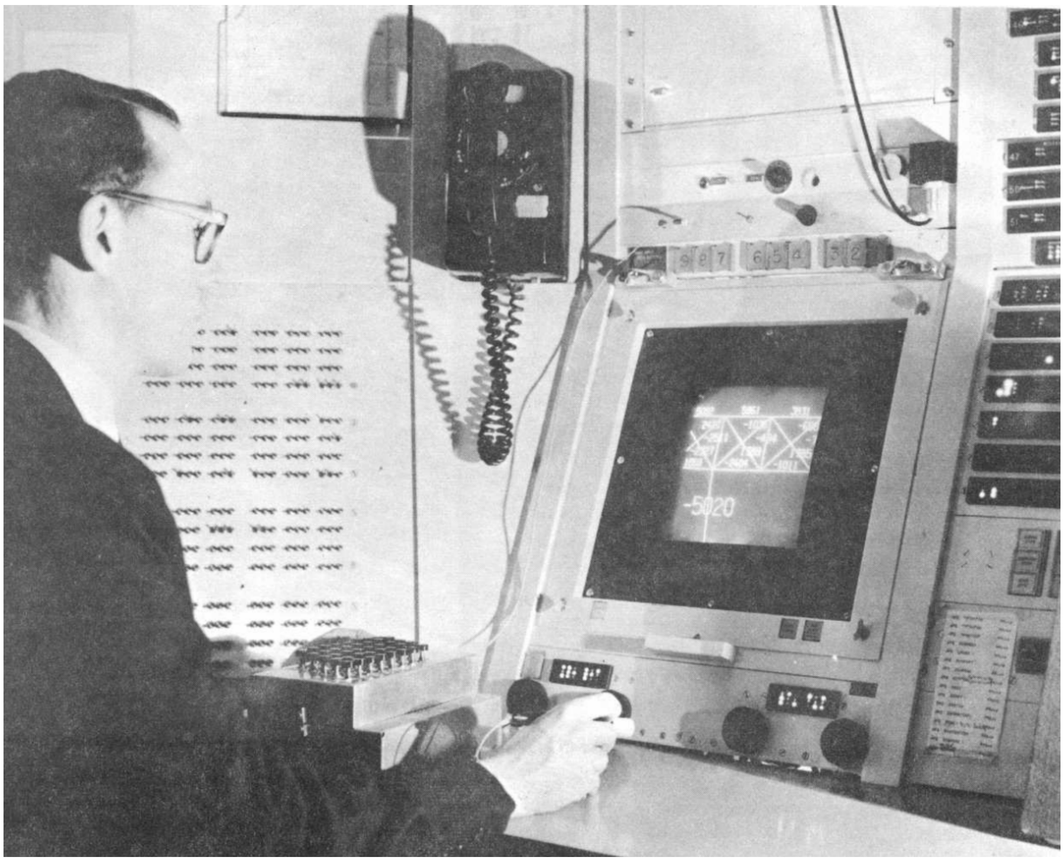


Figure 28.1.: Interacting with Sketchpad [789].

systems. As a result, several prototype systems gradually led to the refinement that is today recognized as the GUI. These technological developments converging with new HCI to form the *direct manipulation* paradigm that we discuss in this chapter.

28.1. A Brief History of the GUI

Limitations in command-line interfaces have long been recognized. The first prototype that resembles our modern GUI is *Sketchpad* [789]. Figure 28.1 shows Sketchpad. Sketchpad demonstrated a series of features associated with a modern GUI, such as the use of a pointing device (lightpen) for drawing geometric shapes, and the ability to create and manipulate graphical objects presented on display. In software code, it used concepts like 'objects' and 'occurrences', which precedes object-oriented programming.

Around the same time, novel systems such as *PLATO* (Programmed Logic for Automatic Teaching Operations) and *HES* (Hypertext Editing System) did not provide a GUI but explored elements central to later developments. PLATO allowed graphics to be shown along with text on a raster display to provide an interactive learning environment and

HES allowed a user to use a lightpen to navigate hypertext (see Chapter 24).

In 1972 Alan Kay set out his vision and requirements for what he dubbed *Dynabook—a personal computer for children of all ages* [400]. Essentially, it predicted a modern laptop or tablet along with a modern GUI, although the target audience for this device would be children.

In 1973, the *Xerox Alto* became the first computer specifically designed to support a GUI. The users could interact with a keyboard and mouse, and the GUI was shown on a black & white CRT raster display. It provided a WYSIWYG (What You See Is What You Get) word processor capable of mixing several font families and font sizes, a graphics editor that allowed the user to change tools by selecting their corresponding items on a toolbar, an interactive SmallTalk environment, and a pinball game, among other applications.

This initial workstation design and its GUI concept were gradually refined (see Figure 28.2) and led to the development of *personal computers*—computer systems that are feasible for individual use rather than relying on a mainframe. This led to the Apple Lisa, the IBM PC, UNIX terminals, and the Windows operating system for IBM PC-compatible computers, which all provided a GUI for users to interact with. Since such computer systems were prohibitively expensive at the time, several personal computers targeted toward consumers, such as the Commodore C64, Amiga 500, Atari ST, and many others, were developed and commercialized in the early to mid-1980s. By the mid-1980s, the GUI was established as the de facto standard user interface, typically complemented by a command-line terminal interface for tasks that could not easily be carried out using the GUI.

28.2. Design Objectives

GUIs exhibit great flexibility in how they can be designed. However, this expressivity is also a challenge. Unlike, for example, a form, a GUI can have many more possible design variants. To aid GUI designers, GUI software libraries' providers make guidelines available, such as Apple's *Human Interface Guidelines*. These and other vendor-provided guidelines are specific to particular products and brands. However, there is general agreement on a core set of objectives that GUIs should strive to meet.

28.2.1. Visibility

GUIs should strive to have high *visibility*. This means that to support users in achieving their goals, these goals and the necessary steps to achieve the goals, should be visible to users.

What You See Is What You Get One example of this principle is WYSIWYG—What You See Is What You Get—word processors, which show the effects of choices of font families, sizes, and styles directly in the document. Headings, body text, hyperlinks, and other formatting elements of the text are shown directly to the user, and the user can

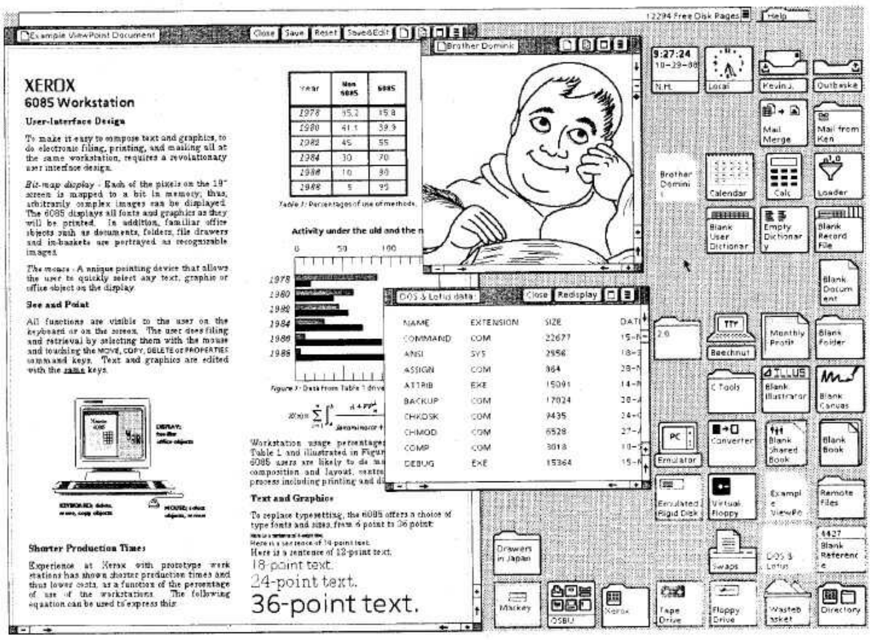


Figure 28.2.: Xerox Star's WIMP interface: Windows, Icons, Menus, and Pointing devices.

interact with these elements directly. For example, if the user selects text and changes the formatting then this change will be immediately visible in the current view of the document. Further, the Font dropdown list shows the font as it will appear on the document after selection.

Visibility of Commands Another example of this principle is to ensure that commands are visible. For example, **??** shows a word processing application that supports the visibility of commands by revealing many relevant text formatting options and tools in the toolbar to the right of the text document. For example, the choice of font, style, size, and color can be set directly. Common text features, such as the choice of having the text aligned to the left, right, center or having it justified, are also immediately visible. If the user does not know what an option means, the user can hover the mouse cursor above an element in the toolbar to trigger a *tooltip* description.

Visibility can be supported by allowing users to browse commands using linear pull-down menus or by exploring a series of toolbars. However, visibility is about more than just making commands and other interaction possibilities obvious to the user. Visibility means designing a GUI that invites users to take action and allows users to easily interpret what the actions are and their consequences. At the same time, both screen real-estate and users' visual attention are limited resources. Therefore, there is a need to strike a balance between how much information that can be conveyed in the user interface without overloading the user.

Visibility of Status Visibility of status means ensuring that the GUI shows the status of entities relevant to the user's goals. An example of visibility of status is an indicator of whether a remote user is available or busy in a communication program. Another subtle visibility of status example is the padlock icon on the web address input field, which indicates that the currently visited website uses a verified certificate.

Visibility of Dependencies An example of a well-established graphical user interface paradigm that attempts to strike this balance is the spreadsheet. Spreadsheets allow users to link cells together by formulas which refer to the values in other cells. Such dependencies can be difficult for the user to understand. As a result, a spreadsheet GUI provides a mechanism for visualizing such dependencies. When the user clicks a cell with a formula that refers to other cells, the dependent cells are color coded to match the color coding of their role in the original formula.

Paper Example 28.2.1 : A graphical user interface with physics

A graphical user interface does not have to look and feel like a traditional desktop GUI. For example, a GUI can be driven by lightweight physics and relax some of the rather rigid organizational principles in traditional. Agarawala and Balakrishnan [12] do this by allowing users to organize files as piles of documents that can be quickly moved by dragging and flicking them with a pen. The resulting *BumpTop* prototype enables windows, photos, and files to be organized informally in piles.

BumpTop supports a range of interaction techniques to select and browse objects. The movement of objects is physics-based and employs rigid body dynamics, friction, and collision detection which means when objects collide they experience semi-realistic displacement effects.

28.2.2. Consistency

To support users in learning a user interface, *consistency* is an important design objective. Consistency means that representations are similar across the user interface. If the print command is represented with a icon showing a particular printer in one part of the interface, the same icon should be used in another. Similarly for interaction techniques. If one has to right-click an object to invoke a particular command, one should use the same right-click for other commands. Consistency might be followed across user interfaces (so-called external consistency). The rationale is that external consistency help users transfer skills from other interfaces. There are several techniques for ensuring consistency. Adhering to style guides—such as Apple's Human Interface Guidelines—ensures consistency across applications on a platform. There has also been developed tools that can help check the consistency of bottom placement and labels [495].

Is striving for consistency always a good thing? There is no clearcut answer. Grudin [294] provides several examples of interface-design decisions where consistency is not attractive. Grudin points out that there are several considerations before employing consistency as a design guideline: 1) consistency must be defined; 2) there is a need to identify good—as opposed to poor—consistency, which means there must be a method of identifying the most suitable consistency among several competing consistencies; and 3) there must be a method of identifying when other principles are more important than consistency. Grudin argues that striving for consistency is often not a useful principle. Instead, interface objects are better placed according to the need of the user's tasks. This changes the focus from properties of the user interface to task analysis and an understanding of the work context.

28.2.3. Minimizing Errors

Errors are outcomes in a graphical user interface that are inaccurate or incorrect. It is important to realize that errors to some extent are unavoidable and even intrinsic in human-computer interaction. For example, noise in human neuromuscular system will make it exceedingly difficult for a user to move a mouse cursor to precise pixel location on a high-resolution display. In practice, users are also bound to make mistakes due to lack of attention, training, or a misunderstanding of the user interface. Further, GUIs can induce errors by relying a confusing design. It is therefore natural to introduce the objective *minimizing errors.* A GUI should be designed to anticipate and tolerate user errors and minimize erronous outcomes. Given the importance in eliminating as many errors as possible, user interface designers commonly use several well-established tactics.

Preempting Errors GUIs should adhere to sound GUI design principles, such as those mentioned in this section. In particular, a clear, simple structure and a user interface that presents graphical elements in a consistent manner can help preempt errors to begin with.

Reversing Outcomes GUIs allow users to reverse the results if their actions are later found to be undesirable. This realization has led to the widespread implementation of *undo* and *redo*, which allow users to reverse their actions (undo) or repeat them (redo).

A simple linear undo/redo facility can be achieved by using a stack data structure: Every last executed command is put on the stack. When the user undos, the last command on the stack is retrieved, and its consequence is eliminated. For example, if the user underlined some text, then the last executed command would be to underline some specific text in the document. Once the command has been executed, the command is added to the undo stack. If the user now wishes to undo this action, this last executed command is retrieved and the consequence reversed (such as removing the underlining from the specific text in the document). This type of undo action can be repeated by the user or reversed if the user wishes to redo the action. The size of the data structure is known as the *undo history.* An application using a linear stack-based undo model can only reverse command execution up to its undo history. More sophisticated undo mechanisms may be required for more complex applications.

Explaining Outcomes and Confirming Actions An established GUI practice is to seek confirmation when the user desires to issue commands that are irreversible or may have unforeseen consequences. Such confirmation can, for example, be implemented by triggering a Dialog Box that explains the consequences of the action and provides the user with options, such as proceed, abort, or retrieve information about the consequences of the action. A common example of this type of dialog box confirmation is when a user wants to overwrite an existing file.

28.2.4. Accessibility

Finally, GUIs should be *accessible* to all users. What this means and how this may be addressed is discussed in the chapter on tool use (Chapter 19).

Paper Example 28.2.2 : Common GUI guidelines

Task support : GUIs should prioritize support for tasks that are frequent, important, or risky. Moreover, their design should offer a structure that is compatible with the task flows that users find natural. In Chapter 15 we discussed task analysis.

Simplicity : ''Keep it simple, stupid" (KISS). Offer only those features that are needed – and nothing else. Adding functionality, graphics, descriptions, widgets, options, and colors can only increase users' workload. Increased workload, in turn, increases stress and decreases performance and satisfaction. Examine the elements of the UI and ask which ones are really needed.

Responsiveness : GUIs should be responsive. Simple interactions like manipulations of widgets should have responses within a few tens of milliseconds. Full views should not take more than a second to load up. Most web pages take 3-4 seconds, however.

Consistency : Users are not *tabula rasa*. They approach your design with *priors*. In other words, based prior exposure to related UIs, they have formed *expectations* about your design's structure and behavior. *Consistency* refers to perceptual similarity between a target UI and prior UIs a user has used. Consistency also pertains to interactions: Actions that have the same consequences should be presented similarly. Consistency makes it easier for users to find elements in an interface and control them. It lowers the need for learning.

Recoverability : Users make errors. Because there is a non-zero probability of an error happening, it is important that the interface minimizes their negative consequences on the user. Does your interface allow users to undo, veto, or recall their actions?

Assistance : Do not ask users to do things that can be done automatically for them. Assist them whereever possible. Information that a user has given earlier should be offered as default options, unless there is a reason that this may not be the likely option. Asisstance increases efficiency.

Scaffolding : Offer support for developing skills. Starting with a new UI is often cognitively taxing and stressful. Over time, users pick up skills for performing the minimum tasks. The UI should assist novices to figure out the interface, for example via tips. But it should not stop there. intermediate-level should be helped to figure out how to increase performance (e.g., by power user features like shortcuts).

Ability-based design : GUI design should support users with different motor, perceptual, and cognitive abilities.

28.3. The Principle of Direct Manipulation

A central principle for graphical user interfaces is the principle of *direct manipulation*. The original definition of direct manipulation was provided by Shneiderman [745] and states that a direct manipulation interface has the following three properties:

- Visibility of the objects and actions of interest
- Rapid, reversible, incremental actions
- Replacement of typed commands by a pointing action on the object of interest

These properties were later refined into the following three principles [752]:

1. Continuous representations of the objects and actions of interest with meaningful visual metaphors
2. Physical actions or presses of labeled buttons, instead of complex syntax
3. Rapid, reversible, incremental actions whose effects are on the objects of interest are visible immediately.

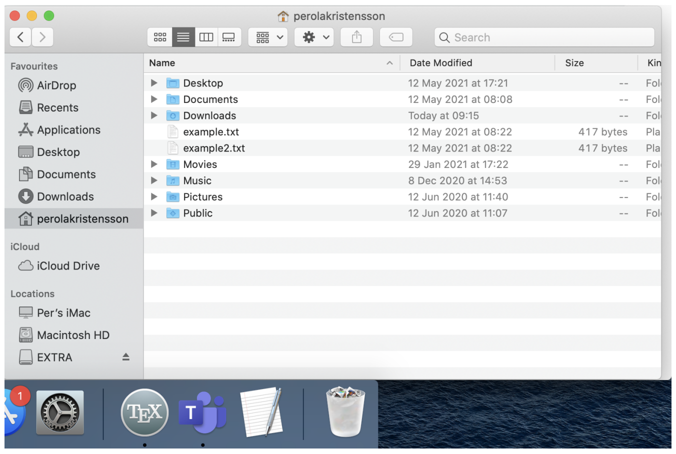
An example of direct manipulation is the removal of a file (Figure 28.3). To delete a file from a desktop GUI, such as Mac OS X, the user opens a file management application window. This window will list all files in the current directory as graphical representations, such as icons coupled with filenames. The user can now move the mouse pointer over a graphical representation of a file and select it by pressing a mouse button. The graphical representation of the file changes in response to this action to provide feedback to the user that this file has now been selected. If the user keeps the mouse button pressed the user can now drag the graphical representation of the file to a graphical representation of a delete action—the bin icon in the lower-left corner in Figure 28.3. When the graphical representation of the file is on top of the bin icon, the representation of the bin changes to provide feedback that the graphical representation of the file is interacting with the bin icon. If the user lifts up the mouse button now, then the file is moved to the bin, which means it is considered deleted but can still be recovered if the user changes their mind. If the user continues to press the mouse button, the system will open a new file management application window.

28.3.1. Benefits and limits of direct manipulation

What makes interaction 'direct'? Hutchins et al. [358] explain directness under two dimensions:

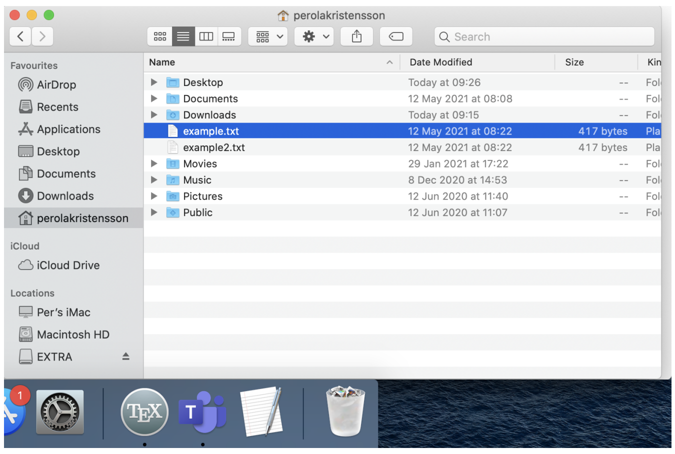
Directness = Distance + Engagement

Distance refers to mental effort required to translate goals into actions and then evaluate their effects. Norman [582] has named these as the gulfs of execution and evaluation (see Chapter 18). *Engagement* refers to the locus of control within the system. Users



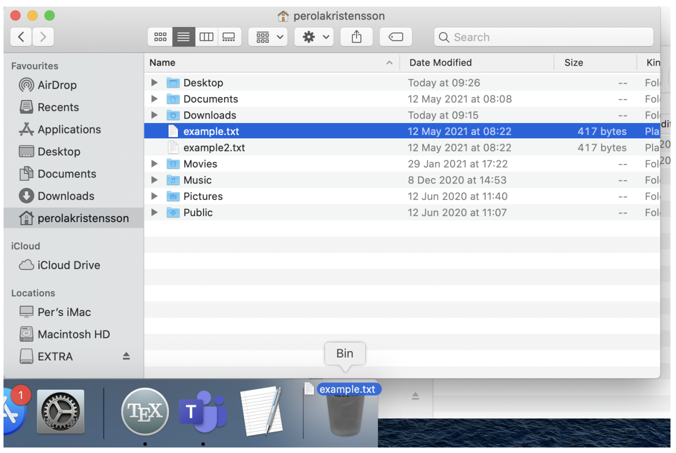
Name	Date Modified	Size	Kin
Desktop	12 May 2021 at 17:21	--	Fol
Documents	12 May 2021 at 08:08	--	Fol
Downloads	Today at 09:15	--	Fol
example.txt	12 May 2021 at 08:22	417 bytes	Pla
example2.txt	12 May 2021 at 08:22	417 bytes	Pla
Movies	29 Jan 2021 at 17:22	--	Fol
Music	8 Dec 2020 at 14:53	--	Fol
Pictures	12 Jun 2020 at 11:40	--	Fol
Public	12 Jun 2020 at 11:07	--	Fol

a)



Name	Date Modified	Size	Kin
Desktop	Today at 09:26	--	Fol
Documents	12 May 2021 at 08:08	--	Fol
Downloads	Today at 09:15	--	Fol
example.txt	12 May 2021 at 08:22	417 bytes	Pla
example2.txt	12 May 2021 at 08:22	417 bytes	Pla
Movies	29 Jan 2021 at 17:22	--	Fol
Music	8 Dec 2020 at 14:53	--	Fol
Pictures	12 Jun 2020 at 11:40	--	Fol
Public	12 Jun 2020 at 11:07	--	Fol

b)



Name	Date Modified	Size	Kin
Desktop	Today at 09:26	--	Fol
Documents	12 May 2021 at 08:08	--	Fol
Downloads	Today at 09:15	--	Fol
example.txt	12 May 2021 at 08:22	417 bytes	Pla
example2.txt	12 May 2021 at 08:22	417 bytes	Pla
Movies	29 Jan 2021 at 17:22	--	Fol
Music	8 Dec 2020 at 14:53	--	Fol
Pictures	12 Jun 2020 at 11:40	--	Fol
Public	12 Jun 2020 at 11:07	--	Fol

c)

Figure 28.3.: An example of direct manipulation. a) Files are visualized as icons coupled with filenames in a file management application window. b) The user selects a file by moving the mouse pointer over the file and pushing down a mouse button. The visual depiction of the file changes in response to this action. c) While holding the mouse button down the user has dragged the file to the bin—an icon that is always shown on the desktop toolbar at the bottom of the screen. If the user lifts up the mouse button at this moment in time the file has been moved to the bin and is considered deleted (but recoverable) by the system.

should feel like being the agents within the system. By contrast, in command language interfaces, the user interact with a hidden intermediary that relays the commands to a third party for execution. This interaction is like shouting a command to someone in another room and waiting for the person to respond. Direct manipulation interfaces offer a *model world*. Think about the desktop metaphor, for example. It models the desktop, even some reduced form of physics. Model-world interfaces allow users themselves to act in the world – in first preson – and not via an intermediary.

Lower distance carries several cognitive benefits over command language interfaces [405]. Using direct manipulation interfaces usually involve recognition rather than recall. Visual recognition is less effortful than memorization of commands. Instead of recalling command names, objects can be searched for visually. Direct manipulation UIs also allow concrete metaphors (e.g., desktop metaphor) as opposed to abstract concepts. Instead of keyboard commands, a pointing device can be used.

However, whether these benefits realize depends on a number of factors. Direct manipulation is not a panacea, but requires a well-crafted GUI design. If the metaphors or icons are poorly designed, or poorly organized, visual search will become a bottleneck. Pointing can also be slower than typing when the GUI is disorganized. Frohlich [256] reviewed empirical evaluations of direct manipulation interfaces, concluding that the evidence is mixed. Some studies show benefits, some not. It is the interface design and the user's task together that determine whether the benefits of Distance can be realized.

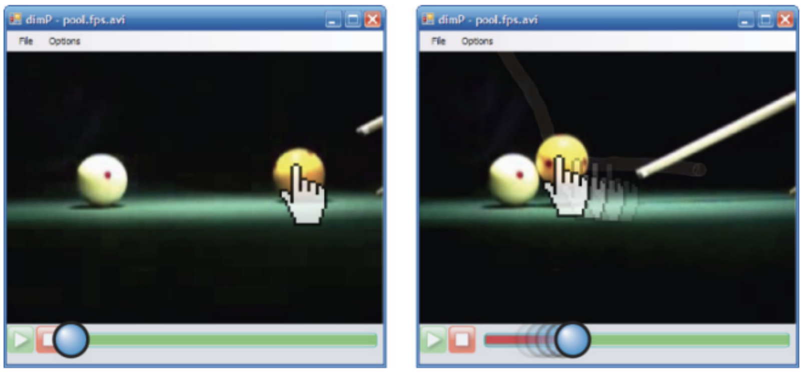
Kieras et al. [405] compared keypad and touchscreen as an interface in a tactical aircraft game. Users were required to identify and act on aircraft appearing on display. They conclude that *response selection* separates direct manipulation and keypad-based interaction. When using a keypad, users must recall which key commands to use in which situation. Direct manipulation lowers this requirement. The authors note that the touchscreen interface would have been even better if less hand movement was involved. However, even small changes to the task or the user interface could have changed this result.

Certain tasks are more difficult for direct manipulation interfaces [255]. It is harder to refer to previous actions, something that can be alleviated with undo/redo and interaction histories. It is also difficult to perform repetitive actions, which in command language interfaces can be done with scripting. Concurrent actions are also difficult because there is a single pointer only.

Paper Example 28.3.1 : Video browsing by direct manipulation

Direct manipulation is a powerful principle for graphical user interfaces. At first sight, however, it may seem limited to the manipulation of icons. However, this is an underestimation of the power of direct manipulation. Let us consider, for instance, whether the browsing of a video could be done using direct manipulation. Would that be possible?

Typically, video browsing is done with a slider that can be dragged across the length of the duration of a video (see the figure below, at the buttom). Dragicevic et al. [209], however, thought that we often have an object of interest in the video, such as a car moving or a ball jumping or our favorite actor doing a particular move. Rather than scrubbing a timeline, could we just grab and move the objects of interest? As shown in the figure below, Dragicevic et al. [209] found a way to make this possible so that video can be navigated as a direct manipulation.



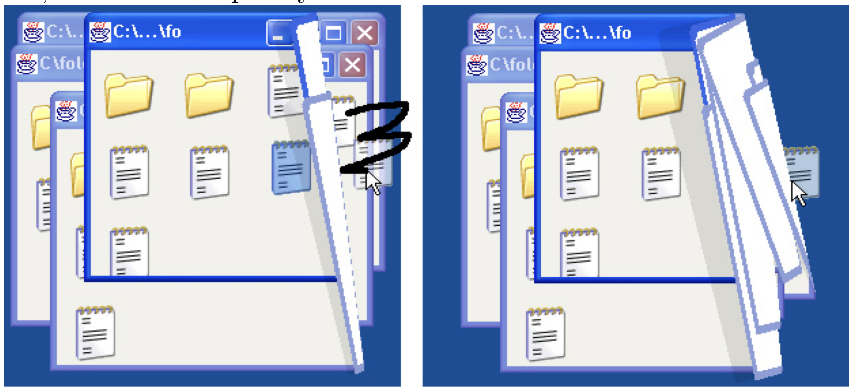
28.4. Anatomy of a Graphical User Interface

28.4.1. Graphical User Interface Elements

Windows A *window* is a central component of a GUI. A window is a (typically) rectangular panel that encapsulates an application or a task. The application windows encapsulate the primary application window. A *window manager* provided by the operating system is responsible for handling fundamental actions for windows, such as 1) instructing the window to redraw its contents; 2) sending keyboard and mouse events to the window when it is in focus; 3) providing user functions for minimizing, maximimizing, and closing windows; and 4) allowing the user to drag and resize windows. **??** shows an example of an application window.

Paper Example 28.4.1 : Folding windows

Windows allows one to expand document and applications into a controllable space on a display. However, as an interactive widget, present-day windows are rectangles that can be moved, resized, and closed. What if the desktop metaphor was extended to windows? Dragicevic [208] presented an interaction technique for windows that allows them to fold like pieces of paper. When multiple windows are on top of each other, the user can quickly fold their corners to see what is underneath them.



Panels A *panel*, sometimes referred to as a *component* or *container*, is a rectangular area that can contain its own background (often simply a solid color) and a set of elements of the user interface. An arrangement of panels is used to create an application GUI. For example, **??** shows a word-processing application window which internally consists of three panels: (1) static toolbar panel at the top providing access to several commands; (2) a context-sensitive toolbar panel to the right which changes depending on what type of content the user has selected (text, tables, etc.); and (3) the main word processor panel.

Menus Menus are widely used in GUIs. In desktop GUIs, linear pull-down menus allow the user to traverse a menu hierarchy to explore and select commands. On a mobile touch-enabled device, this menu exploration style is impractical and therefore menus are usually shown as a series of horizontal buttons that the user can navigate. Menus are discussed in more detail in the chapter on interaction techniques (Chapter 26).

Icons An *icon* is a graphical depiction that represents some capability of the GUI, such as the ability to launch an application or trigger a command. Icons can be generated by bitmaps or vector graphics. Depending on context, icons can serve a number of purposes. First, an icon can be selectable by the user. For example, a group of icons representing an application or file can typically be selected by either selecting them in sequence. This is typically done by holding down a specific keyboard key while using the mouse to click on each icon). Alternatively, it is sometimes also possible to use a mouse and perform a rectangular selection. This allows a user to perform an action on multiple targets simultaneously. Second, icons can be clicked. Such an action typically triggers an action, such as the launch of an application or a specific command. For example, clicking on an icon representing a pair of scissors usually triggers the command *Cut*. Third, icons can serve as targets for actions. For example, it is often possible to delete a file or application by dragging its icon to the bin icon. Fourth, icons can reveal information. For example, the bin icon may be represented as an empty bin when there are no recoverable files available and as a bin with contents when there are recoverable files. To access the recoverable files, the user can click the bin icon, which opens a file explorer window showing the special system directory that contains deleted but still recoverable files on the system. Another example is app icons on a smartphone, which may show a number next to them to indicate the number of unseen notifications they can reveal to the user if the user triggers the app icon by touching it.

28.5. Designing a Graphical User Interface

GUI design is a so-called Easy/Hard activity. It appears easy. Anybody with moderate computer skills can just 'wing it': download an interactive SDK (software development kit) and start drawing a GUI. There are plenty of examples online to learn from. However, designing an actually usable GUI is hard. You are almost certainly going to fail your users unless you follow a systematic design process or happen to be a design veteran.

The user-centered design of a GUI is not different from other areas of design, yet some unique characteristics exist. First, GUI designs are based on user research. Key decisions in GUI design - like those concerning functionality, UI style, visual design approach - require empirical understanding of the users' practices, needs, capabilities, contexts, terminals, existing designs, and so on. Designers increasingly rely on logs from existing applications as a source of data. Second, design is iterative. Hundreds or thousands of alternatives are sketched, eventually converging towards hi-fi prototypes worthy of evaluation. Digital sketching and wireframing tools are often used to boost productivity. A *UI kit* is a template of components that can be used in design tools like Sketch and Figma. Several iterative cycles or sprints are carried out that often involve some research, design, and evaluation. Third, GUI designs are evaluated with dedicated methods. Analytical evaluation methods (e.g., GUI design guidelines) are used early on in the project and empirical methods like usability testing, AB testing, and field studies later on. Elsewhere in this book, we have discussed heuristic evaluation guidelines for GUI design (see **??**). GUI designs need to be reviewed against company-specific guidelines and design systems. For example, at the time of writing, Microsoft offered *Fluent Design System* and Google *Material Design*. In a review or audit session, a design team reviews all low- and high-level details of the candidate design. Changes to the UI after this point will be very expensive.

Development of GUI software is still a significant cost. Although there are powerful SDKs, and even sketch-to-code solutions, developing a full-functional GUI that fulfills the specifications is time-consuming. The software controlling the GUI must be interfaced with application features, business processes (e.g., payment), databases, and hardware. Software engineers must validate the software before launch. However, GUI design does not end at the release of a product, but metrics and processes must be put in place to maintain and update it.

In the following, we provide a primer for five elementary GUI design considerations: visual layouts, text and labels, icons, widgets, and metaphors. We review approaches based on empirical research. GUI design, however, is first and foremost a practice. The best way to learn GUI design is to try it out. For more detailed treatments of the topic, we refer the reader to dedicated practical books [752, 875].

28.5.1. Visual Organization

Visual Hierarchy : Elements on a graphical layout should be organized logically. But what does that mean? Visual hierarchies use visual cues to define which elements belong together and in which order they should be looked at (see **??**. Hierarchies become necessary when the number of element grows. Instead of N separate elements, aim toward a hierarchy. Use regular visual cues (e.g., rectangular boxes, colors) to indicate groups. Gestalt laws are useful for visual grouping. Contours, regions, colors, paddings, and margins are strong cues of what elements belong together.

Visual hierarchy ranks graphical elements and places them to larger groups: elements higher in the ranking contain elements on lower levels. A well-designed hierarchy helps the user understand the order in which a task can be completed (task flow). Visual cues like proximity, common region, contiguity, and regularity in colors and style are used to communicate the hierarchy.

Grid line is the most important organizing principle of layouts. Grid lines are horizontal or vertical lines that intersect a UI canvas. They define how the edges of UI elements can be placed. *Grid layouts* are UI layouts defined with grid lines. A grid layout is a spatial structure that places elements in a non-overlapping way on a (typically) rectangular area. A layout can be described efficiently with grid lines. One can easily define repetitive structures like column grids and card grids, but also more complex structures. They are used across various stages of GUI design, from sketching and wireframing to prototyping and deployment

Grid alignment is a measure of GUI complexity (see [**?**]). It describes the minimum number of grid lines that can be used to define a layout such that all edges of elements fall on a grid line. Empirical work on grid alignment has found it to be correlated with perceived complexity of a GUI: The smaller this number, the less complex it is perceived [533]. In a controlled study measuring search time and preference, Parush et al. [624] showed the importance of grid alignment and balance: "The combination of poor alignment and poor local density had the strongest adverse effect on search time. Alignment and grouping were found to have more influence on subjective preference" [p.343]. However, search times shortened considerably with experience, even for the worst designs.

Visual Guidance : GUI design should have a *visual flow* that matches the user's *task flow* (see Chapter 15 for task analysis). Visual flow refers to the order in which users are likely to scan the regions of a UI. By contrast, *perceptual clutter* is a state where everything is competing for the user's attention, and it is not clear where to start a task. Flow is affected by the use of colors, sizes, shapes, and orientations. *Negative* space - that is, the use of white space - can be used to guide attention, too. We learned in Chapter 3 about visual saliency and clutter, which provide a systematic concept for understanding what grabs a user's attention.

28.5.2. Icons

Icon design needs to take into account a number of factors. First, an icon may serve to represent many purposes at once as an icon can be a representation of a digital object, an underlying action, a receivable action, or all. An example of such an overloaded icon is the bin icon. One representation is status: an empty bin communicates to the user that there are no recoverable files available, while a bin with contents signals that there are recoverable files. Another—related—representation is the fact that if the user clicks the bin icon, then the system will open a file explorer window showing any available recoverable files. Further, set of files and applications can be deleted by first selecting them and then dragging their icons to the bin. When this happens the representation of the bin icon changes, for example, by a subtle change in shading, to indicate that the bin icon has been selected. Finally, when files are deleted by dragging them to the bin the system plays a sound representing someone crumbling a piece of paper as an indication to the user that the action was carried out.

Second, the design of an icon must be understandable by the user. This may involve choosing a metaphor the user is likely to understand, such as the bin icon design. As another example, a notepad app icon may literally depict a notepad. Application icon design often foregoes the idea of using metaphors and instead opts for ensuring the icon represents the brand of the application in question. For example, application icons, for Microsoft Word, Skype, etc. all relect their respective brands.

28.5.3. Metaphors

To guide interface design it may be possible to leverage *metaphors*. The idea behind the use of metaphors is to design concepts, processes and actions in a graphical user interface around the user's prior knowledge, such as designing digital office tasks, for example file management, note taking, word processing and so on around a desktop metaphor. Figure 28.4 shows an example of an aspect of this desktop metaphor in the form of a bin serving as a representation for for discarding unwanted files and applications by throwing them in the bin and, conversely, recovering such discarded files and applications by picking them up from the bin before they have been thrown out with the trash (deleted permanently).

Carroll et al. [134] lists several such *interface metaphors*. Word and text processing uses the metaphor of the typewriter and exploits prior knowledge around using typewriters, typing paper and keyboards. Desktop publishing and document structures (such as headings and subheadings) uses the metaphor of a physical document and exploits prior knowledge around typesetting, inserting physical figures and the logical structure of a document. Single-user or collaborative electronic workspaces use the metaphor of a chalkboard or whiteboard and exploit prior knowledge around group interaction around a chalkboard or whiteboard and the free-form organization of text and graphics. Desktop

Figure 28.4.: An example a commonplace metaphor in GUI desktops: the bin as a representation of deleting files.This is the Mac OS X bin. Users can delete files by dragging them to the bin (or issuing a delete command directly). By clicking the bin the file management application shows the user previously deleted files, which can then be recovered if the user so chooses.

accessories use metaphors of desktop tools and exploit prior knowledge around the use of a calculator, notepad and personal organizers. Further examples, include spreadsheets exploiting prior knowledge of ledger sheets for understanding matrix-structured data organization and forms-based applications exploiting prior knowledge of business forms and the codification of existing business processes into forms, organization of information, and report generation.

The central idea behind interface metaphors is to support learning by analogy by exploiting users' prior knowledge in the design of a graphical user interface to able to transplant existing concepts, actions and processes the user is already familiar with to improve users' learning and understanding of graphical user interfaces. However, it is critical to realize that metaphors are far from perfect mappings to user interfaces. In fact, as noted by Carroll et al. [134], if metaphors provided a complete mapping then, for example, a word processor would actually fully appear and behave as an actual typewriter.

As a result, the mapping between the source of the metaphor and the target (the user interface) is never one-to-one and this results in *metaphor mismatches*. For example, a forms-based user interface may present an electronic form similar to an existing physical business form. However, an electronic form can, and probably will, support input validation and can thus provide feedback to the user on invalid input, suggest probably inputs, and also prevent the user from assigning the next part of the form until the first part of the form has been fully validated.

Further, interface metaphors are often used in combination which leads to *composite metaphors* [134]. For example, a desktop user interface uses a desktop metaphor while desktop accessories use their own metaphors, such as notepads, calculators, clocks and personal organizers.

28.6. Why do we still have GUIs?

As explained at the beginning of this chapter, research on GUIs begun in the 1960 and the first commercially successful GUIs date to the 1980s. As pointed out by Beaudouin-Lafon [56], this is curious because all other aspects of computers have improved dramatically.

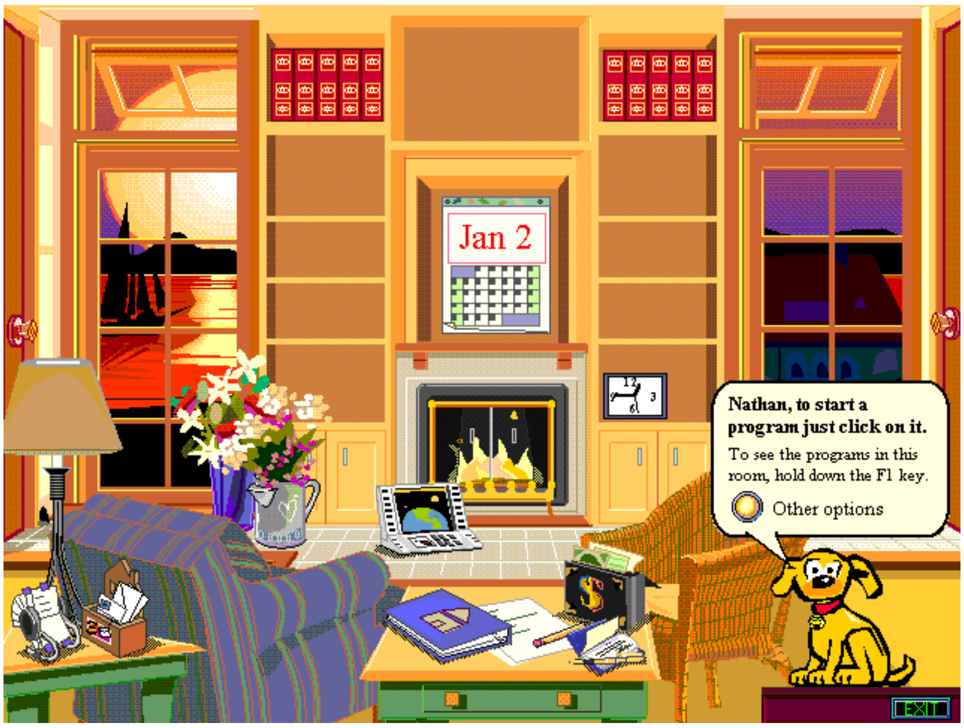

Jan 2
Nathan, to start a program just click on it.
To see the programs in this room, hold down the F1 key.
Other options
EXIT

Figure 28.5.: An example of a GUI making extensive use of metaphors: the Microsoft Bob product for Windows 3.1. Bob was a proposed graphical shell for inexperienced computer users. The user can start applications by clicking on their representations in the GUI. For example, the side table to the left of the armchair shows an address binder to the left and a mail tray to the right. Clicking the address binder will start the address book application and clicking the mail tray with start the email client. Figure source: http://toastytech.com/guis/bob.html.

The Intel 8088 from 1979 ran at 5MHz, while current CPUs run at 5GHz with multiple cores and are capable of about 400,000 as many instructions per second. So how come that the user interface, the GUI, has remained the same?

One answer is that the GUI is a fundamentally outstanding idea. Using space to structure information helps see what one can do and aids memory. Direct manipulation has made trial and error available to most users in a way where they can back up. Visual representations remain compelling for many users.

Another reason is that, in reality, GUIs are being combined with all kind of other UIs. Thus, what seems to be a dominant interface style is increasingly a mix of graphical elements, search interfaces, keyboard commands, and other interface styles. Consider for instance the use of speech when using mobile devices: Many users make calls or start applications in this way, rather than using the GUI. Another example is the use of commands in a desktop operating system: Here, many users switch to a command-like interface for search, for handling files, or for shortcuts to applications. Thus, the GUI is increasingly combined with other forms of interface.

A third reason is that the alternatives to GUIs are wanting. Interfaces that are based on naming or searching for commands are sometimes hard or impossible to use for some tasks. Interfaces that automatically infer what we want, rather than take commands like the GUI, have been a vision for decades [573]. Intelligent agents and brain-computer interfaces are two examples. The former, however, has been described as having "a really bad personal assistant" [483] and the latter is currently only feasible for rather simple tasks (movement, simple choices). In Chapter 29 we discuss some other alternatives to GUIs that show promise.

Summary

- Graphical user interfaces were invented more than 40 years ago but is still the main interaction style in many computer systems.
- Direct manipulation is a key principle of graphical user interfaces.
- Graphical user interfaces exploit a number of visual design principles, including visual hierarchy, grid lines, and undo.

Exercises

1. Visual design principles. Analyze the visual design principles followed in the design of a GUI you are familiar with. Take a screenshot of screen you use commonly and annotate it with principles we listed in this chapter.
2. Direct manipulation. Consider a word processor you are familiar with. Explain how it uses direct manipulation in common tasks. Now consider how you would redesign the word processor to *not* follow the principle of direct manipulation.

3. GUI design. Following the design principles outlined in this section, design a simple GUI for a note-taking app for two devices: a laptop computer and a mobile phone. Note down design challenges that differentiate the two terminals.

4. Icon design. Some icons are based on legacy technology that are not familiar to younger generations, such as the save icon (floppy disk), phone call (phone handle), folder (file cabinet), address book (rolodex). Following guidance given in this chapter, design new icons (maximum size 300 x 300 pixels) for these.

5. Widget design. Consider the first instance of a physical table and write down the attributes of this table that first come to mind.
   a) Now ask a friend to do the same. How many attributes are similar?
   b) Now consider a typical table widget in a GUI (see e.g., spreadsheet applications). How many attributes does the table widget in the GUI share with the attributes you previously arrived at?

6. Cognition and GUIs. Consider the three principles of direct manipulation. Which insights on understanding people from Part II justify those principles? For each of the principles, come up with three insights.

29. Reality-Based Interaction

The user interfaces for computer systems have remained remarkably consistent since the emergence of the graphical user interface in the 1980s. Processing speed has increased dramatically, storage is abundant, and network speeds are orders of magnitude higher today. Therefore, it is astonishing that we still rely on the same input methods, notably keyboards and mice, and the same user interface paradigm, including windows, icons, menus, and pointers [55].

The interaction paradigm that originates with the desktop has several limitations. First, the user is assumed to be stationary and the physical surroundings are not changing Second, there is the assumption that only one user interacts with the interactive system at a time. Third, input, at least with desktop and laptop computers, is indirect, using devices such as mice and physical full-size keyboards. Finally, the user's surroundings have no special status in interaction. As a consequence, users do not have a natural way to interact with objects and people close to them.

To address these limitations, researchers have worked on several ways to characterize new opportunities for user interfaces. They have attempted to articulate in a principled way why and how user interfaces should be developed beyond the desktop computer model and its focus on windows, icons, menus, and pointers. These attempts have mostly been negatively worded, such as "post-WIMP" or "beyond the desktop" or "non-command user interfaces".

One vision stands out because it articulates the positive aims and characteristics of this type of user interface, rather than simply stating that we need to go beyond GUIs. This framework is *reality-based interaction*, developed by Jacob et al. [374] (see Figure 29.1). It includes four goals to build interactive technology that better exploits and supports our capabilities.

- Naive physics: "people have common sense knowledge about the physical world" [p.222]. What has been called tangible user interfaces move the control of computers and the display of information into the physical world (see section 29.3).
- Body-awareness and skills: "people have an awareness of their own physical bodies and possess skills for controlling and coordinating their bodies" [p.222]. Interaction in mixed reality is widely based on this idea that you move your limbs as you would do outside mixed reality (see section 29.4).
- Environment awareness and skills: "people have a sense of their surroundings and possess skills for negotiating, manipulating, and navigating within their environment" [p.222]. Interactive systems may use the user's position to display particular information.

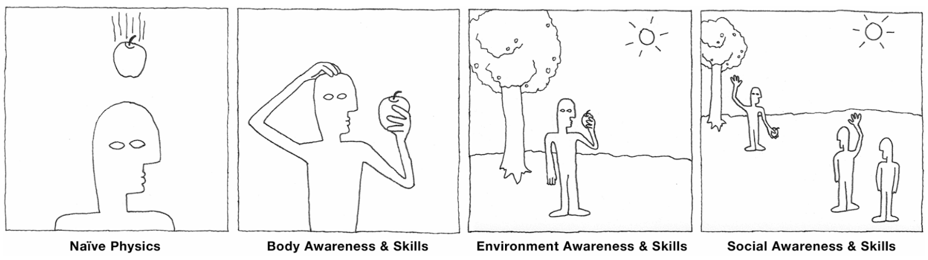


Figure 29.1.: An overview of the four areas in reality-based computing Jacob et al. [374].

- Social awareness and skills: "people are generally aware of others in their environment and have skills to interact with them" [p.222]. An example of attention to such awareness and skills is the interest in video communication systems to get the gaze direction of a remote person to appear correct.

In practice, this vision boils down to research on interfaces that in various ways break key assumptions about desktop-based computing. In this chapter, we will articulate four such types of interface that each break a particular key assumption.

- *Mobile interaction* that breaks the assumption that interaction means that a user is primarily stationary.
- *Multimodal interaction* that breaks the assumption that interaction with computer systems primarily involves input using one modality and output using another, such as a visual display.
- *Ubiquitous computing* that breaks the assumption that users must command computers to get things done. Ubiquitous computing exploits context sensing to enhance and automate interactions for the user.
- *Mixed reality and tangible interaction* that break the assumption that virtual reality and physical reality are separate.

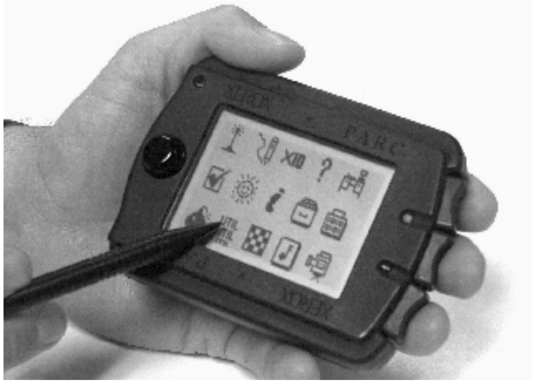
Paper Example 29.0.1 : An early vision of ubiquitous computing

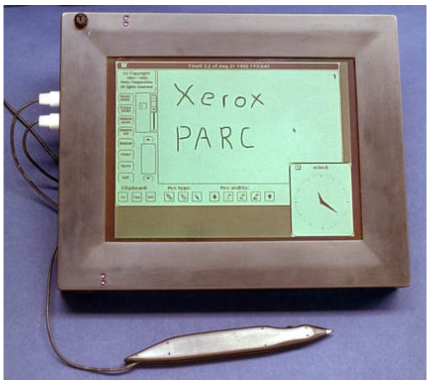
Weiser presented a prominent vision of going beyond desktop computing in a paper in *Scientific American* [860]. Weiser outlined three types of ubicomp system that had been developed by him and his colleagues at Xerox Parc. They envisioned that computing should come in different sizes, each suitable for a particular task. Weiser noted that in any given room, there are hundreds of differently sized writing and display surfaces. He focused on three scales, tabs, pads, and boards and stated:

> Look around you: at the inch scale include wall notes, titles on book spines, labels on controls, thermostats and clocks, as well as small pieces of paper. Depending upon the room you may see more than a hundred tabs, ten or twenty pads, and one or two boards.

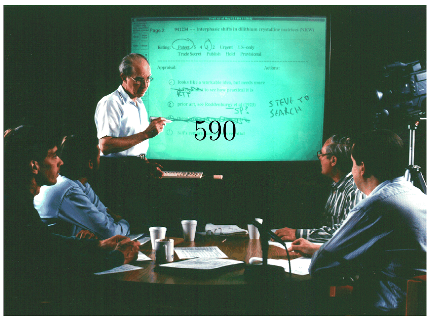
An example of a tab-sized system is ParcTab [853] (below, left). The ParcTab had a 6cm by 4cm monochrome display that could be used with a pen and a dedicated input alphabet. The ParcTab could also be used in just one hand, controlled by three buttons on the side. It was made possible by innovations in low power consumption and networking protocols. The ParcTab was used, for instance, as remote pointers in meetings, for impromptu voting, or for paging people in a manner depending on their context.

The ParcPad is shown below, right (photo used with permission from PARC, part of SRI International). This device was connected to other devices via a radio network and had a stylus with multiple buttons. In contrast to tablets known today, the vision of the Pad was to be a scrap computer that could be picked up and used when needed. Just as few people carry loose, partially used pieces of paper, the ParcTabs were intended to lie around and be grabbed when needed.





An example of the vision of the board was LiveBoard [221]. LiveBoard supported wireless pen input and a back-projected screen about 120 cm times 80 cm. The Tivoli application [630] shown below supported informal meetings around the LiveBoard (photo from [534]). Tivoli supported issuing commands with gestures and allowed multi users to interaction at once. In addition, Tivoli allowed the simultaneous sharing of content with other Tivoli applications at other places.



Those three systems were influential not only in HCI research but in computer science in general, because they were backed by a vision, technically advanced for their time, and extensively used by the researchers themselves.

29.1. Mobile User Interfaces

Smartphones, interactive mobile computers, made their explosive entry into the consumer market almost two decades ago. What few know, however, is that HCI research on the topic was ahead of this development by two decades. The topic was brought to the attention of the research community in a way that was way ahead of the curve at the beginning of the 1980s. In 1983, when the PC (personal computer) was making its way to the consumer market, Norman anticipated the development of computers to a smaller form factor. He extrapolated a trajectory of improvements in computing power and discussed the implications to trade-offs in interaction that designers must face. His keynote at CHI in 1983 [580] [p.9] ended with an insightful speculation about its consequences:

> New developments in technology are moving computer systems in several conflicting directions simultaneously. Workstations are getting more powerful, with large memories, large, high resolution screens, and with very high communication bandwidths. These developments move us toward the ability to present as much information as is needed by the user with little penalty in time, workspace, or even memory space. At the same time, some machines are getting smaller, providing us with briefcase sized and handheld computers. These machines have great virtue because of their portability, but severe limitations in communications speed, memory capacity, and amount of display screen or workspace.

Norman outlined challenges to user interfaces due to the radically different form factor and computational capabilities of the mobile computer:

> Just as workstations are starting to move toward displays capable of 1000 line resolution, showing several entire pages of text, handheld computers move us back toward only a few short lines – perhaps 8 lines of 40 characters each – and communication rates of 30 eps (300 baud). The major differences between workstations and handheld computers relevant to the tradeoffs discussion are in the amount of memory, processor speed and power, communication abilities, availability of extra peripherals, and screen size: in all cases, the handheld machine has sacrificed power for portability. Because the same people may wish to use both handheld machines and workstations (one while at home or travelling, the other at work), the person may wish the same programs to operate on the two machines. However, the interface design must be different, as the tradeoff analyses of this paper show.

Norman made an important observation about the changing form factor. Small size accentuates a trade-off he saw in desktop computing: that between workspace and menu size. Consider a social media application that you use, for example: how much of the screen estate is dedicated to general menu options (e.g., 'Camera', 'Post') versus the space dedicated to reading and editing posts? If you were to add one general menu option, where would you add it? The larger the menu that one needs to navigate an application, the less workspace there would be left for the actual application. Any designer who designs for mobile use would have to solve how to share display space between dynamically changing and static elements.

However, the first technical innovations in this space occurred even earlier than that. E.A. Johnson introduced the capacitive screen in 1969. The patent precedes the touchscreen we use on smartphones today. In 1972, Kay released a concept design of a mobile computer called the Dynabook (see Figure 1.5) [46]. One can appreciate how Dynabook was at that time by considering how computers were used then: via textual terminals that accessed a mainframe computers. Dynabook demonstrated a media-rich networked computer that one could carry. It had a graphical user interface and multimedia capabilities. Although the concept inspired laptop computers, the first working prototype of Dynabook – as it was envisioned – was presented almost 20 years later, largely because the concept was so much ahead of the technology of its time. GRiD Compass was one of the first portable computers, it was a laptop computer presented by Moggridge.

29.1.1. Characteristics of the Mobile User Interface

Compared to the desktop computer, the most radical feature of the mobile computer is its size. Although anyone can see this difference, however, its effect on interaction is subtler and far from trivial. Size affects almost all elements of interaction from the way we select objects to the contexts of use, with significant implications to design.

Size affects the types of input and output that is offered. Commercially available smartphones utilize touchscreens, speech, and physical buttons for input. However, the market has seen a very rich palette of alternatives, many of which have been investigated in HCI research, including radar-based sensing (e.g., Google Soli), computer vision (e.g., eye tracking based on front-facing camera), and the stylus. Output devices are not limited to touchscreens. Mobile computers can use mobile projectors and they can connect to head-mounted displays and glasses, on-body haptic actuators, and even link to other near-by pervasive displays. The modern smartphone also features a wide variety of sensors. These allow inferences to be made about the user's context and to take information into account in interaction. Commonly used sensing include location sensing (e.g., GPS, wireless radio-based methods), gesture and motion sensing (e.g., accelerometer and gyroscope), audio and speech sensing (microphone), and light sensing (e.g., for detecting if the phone is held against the ear). Also more exotic sensors are available, some devices even offer thermal imaging.

The mobile computer takes advantage of the familiar graphical user interface we find on desktop computers. By comparison, design for mobiles generally strives toward having fewer items. Design for mobile interfaces emphasizes *visual hierarchy*: using saliency and containment to communicate the most important areas, a concept which is elaborated in the chapter on graphical user interfaces (Chapter 28).

Because the viewport is smaller, it requires interaction techniques (see Chapter 26) to support elementary interactions, such as translation (scrolling) or paginated (changing page). While collapsible menus (e.g, hamburger menus, which open up side menus when the user clicks on designated icons) and widgets are common, dropdown menus are rare. To save screen estate for the workspace, text entry is not always available, but only in a specific mode. Certain interactions, like hover-overs, are not available. Spreadsheets, which require both workspace and data entry, are avoided on mobiles.

The most distinct characteristic of the mobile device, however, is its portability. Mobile interaction often takes place while users are in transition: walking, commuting, or just multitasking. This poses large and nontrivial challenges to both design and empirical research. The challenge for design is to make interactions usable across the various situations the user may enter. Simply, the more situations a design can support, the more mobile it is. 'Mobile first' refers to the design idea that a user interface that works well in mobile contexts will work well also in desktop settings, and design should therefore start from the mobile device.

29.1.2. Small-screen interaction

The small size of the computer poses non-trivial challenges in interaction. The first is the limited viewport size of the display. Consider navigating a map, for example: the map is larger than what the display can present at any given time (viewport). It needs a offer a way to translate the viewport. Zooming in/out helps the user orient to the map and plan routes.

The other challenge is that fingers occlude the display. If you press a touchscreen with your fingertip, no matter how gently you try to do that, the area that contacts the surface is much larger than a single pixel. So, which pixel did you *intend* to select? An innovative study by Holz and Baudisch asked participants to press a crosshair presented to them [344]. What they did was to look at which mental representation of the fingertip best predicts the recorded touchpoint. They found that the perceived features of the finger – especially its outline and the nail – predict where users intend to touch. Users do not perceive those features from their own viewpoint, but from a hypothesized top-down perspective, like looking down from a 90 degree angle. This is shown in Figure 29.2. This leads to systematic biases in where the finger is pressing. However, because the bias is partially systematic, it can be captured in a model, which can be used to improve the precision of touchscreen sensing in software-side.

The problem of occlusion (by fingers) has been a continuous topic of research in the area of interaction techniques. Many techniques studied in HCI literature are in use in commercial products. *Touch offset* warps the selection point from the inferred touch point to make it easier to select a small target. In text entry, *caret* marks the location to which characters will be inserted. *Caret offset* refers to the offset to the touch location in pixels. *Magnifying glass* is a mode in which the region of the display under the finger is zoomed in in a warped area of the display.

While the mobile interface may appear easy to use to many readers of this book, it is imperative to remember that its small physical size can pose serious accessibility issues. Interacting with small screens is often challenging for aging users and users with impairments. Aging is associated with impairments, such as those related to motor control, strength, and visual acuity. Arthritis and other physiological impairments may cause physical pain in fingers and wrists, making it difficult to operate a mobile device. Loss of visual acuity makes small text illegible. What may appear a completely clear

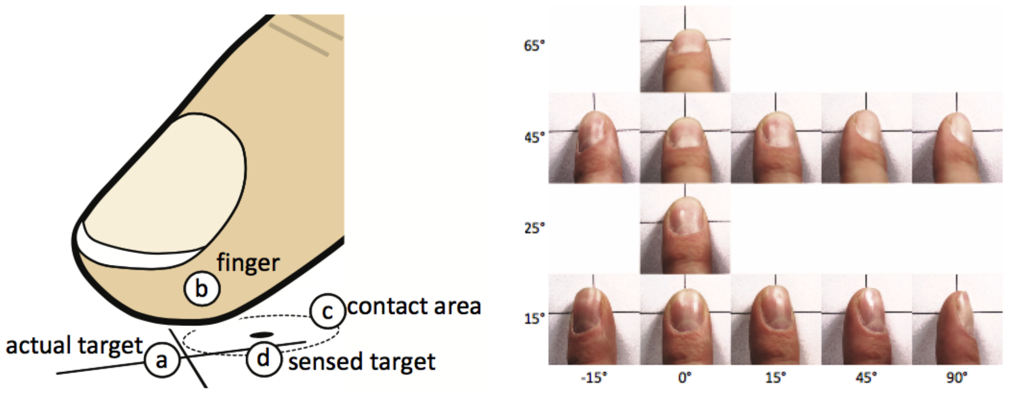


Figure 29.2.: Interactive objects presented on mobile displays are small, how do users select them? A study found that users use the perceived features of the finger – especially its outline and the nail – to touch a target, here marked with crosshair. Users align perceived features in a hypothesized top-down perspective. This leads to systematic biases that can be corrected in software-side with a proper model [344].

button for you can be illegible for others. Given the growing digitalization of services, the ability to operate small screens may accentuate differences between those who can versus cannot participate in the society. Interaction techniques that ameliorate these differences is an open topic in HCI research.

29.1.3. Mobility

Having a computer in your pocket enables new uses for computing that are not constrained physically as desktop interaction is. *Mobility* presents a fundamental new phenomenon as well as opportunities for novel interactions. *Mobile interaction* means support for interaction that takes place while in physical transition between places.

The effect of mobility has been a topic of HCI research for two decades. *Mobile context* refers to factors that are relevant in a situation that may affect interaction with the computer. A defining aspect of mobility is locomotion. *Locomotion* refers to walking or movement by other means. Walking requires visual attention which now needs to be shared between the mobile device and the environment. *Navigation* refers to finding one's way in an environment. This requires not only the ability to localize oneself but also the ability to choose where to go next.

While moving in an environment, users' capability for interaction is degraded. *Environmental noise* refers to physical perturbations – like tremble in a metro – as well as visual noise (e.g., blinking lights) and auditory noise. Such factors can degrade a user's sensori-motor performance. Users also need to move. *Multitasking* is a prevalent characteristic of mobile interaction. Users are always always doing something simultaneously. As we learned in Chapter 5, multitasking requires users to strategically allocate limited cognitive resources among tasks. Multitasking while mobile can be perfectly understood from the same perspective. We share visual attention to events in our environments while using mobiles. We plan and reason about routes when walking, which may distract with planning the mobile device requires. Mobile contexts are also *social*. When mobile, we are constantly positioning ourselves physically and socially among other people. Consider the way we queue, take lanes, or sit: there is an invisible 'order' that we observe, which also affects the way we use our devices. All of these factors make mobile interaction *resource poor*: when designing an interface, we cannot expect visual attention and other resources to be fully deployed to the interface.

Just how 'resource poor' is mobile interaction, what does this mean in practice? In a study called *'Interaction in 4-second bursts'* [609], Oulasvirta and colleagues asked smartphone users to do information retrieval tasks while going through a route in Helsinki. The participants were asked do regular tasks like ordering some coffee, taking subway, and walking through busy streets. The authors set up a wearable minicamera system and recorded how their eyes moved using several cameraviews: one attached to their device, one to their chest, and a third one carried by the experimenter who shadowed them. Their striking finding was that users focus on mobile devices for short bursts of a few seconds at a time. Average glance durations among the contexts were large, from about 4 seconds when walking on a busy street to over 10 seconds when seated. Locomotion and social, dynamic environments were concluded to compete for the user's visual attention.

29.1.4. Habit-formation and addiction

Are mobile computers addictive? Consider having two options right now: the one offers almost instant stimulation and requires almost no effort, and the other requires effort and has its gratifications delayed. Which option would you pick? This thought experiment demonstrates why mobile devices have the potential to form habits. Mobile devices offer us instant gratification with very low cost. Right here right now, it is rational to pick up the device and use it. However, the more you use the device, the less likely there is to be anything new in the news and social media feeds. In other words, mobile device use often has diminishing returns: the more you use, the less you obtain. It is the lure of immediate reward that causes addiction. Problematic use of smartphones has steadily increased the last years. Partially this is thanks to the increasing availability of stimulating contents like games and social media.

However, the picture is more complicated. Statistical methods like structural equation modelling have been used to shed light on factors affecting compulsive user. In a questionnaire-based study of 325 individuals, Yu-Kang Lee et al. [459] found that compulsive usage of smartphones is affected by psychological traits. Users who experience high locus of control, anxiety for social interactions, materialism, and need for touch were more prone to compulsive use. Self-reported gender affected these. Recent research has looked into 'detoxication' applications and theory-based intervention programs to help users overcome problematic behaviors.

29.1.5. Discussion

A significant change took place in HCI's empirical methodology around 2000s when it shifted focus to mobile interaction: a change from laboratory conditions and controlled experiments to field studies [413]. Although obstacle courses and treadmills can be used to simulate mobility in the lab, field studies are argued to offer a more generalizable view to interaction, in particular by allowing everyday events to be included in observations. Mobility itself, and the different contexts it entails, entail a plethora of events and circumstances pratically impossible to stage in laboratory conditions.

Field studies of mobile use have exposed two major phenomena associated with mobile devices. First, multitasking is a defining characteristic of mobile interaction. Users who multitask while driving cause a risk to safety. Mobile devices are associated with 25 % of collisions in the US [29]. Second, the constant availability of mobile devices also distracts our regular patterns in life, and is associated with disturbances in sleeping habits and in gaming addiction.

29.2. Ubiquitous Computing

The term *ubiquitous computing*, or ubicomp, was coined by Wieser [1991] to identify the emergence of a new form of computing. The box above describes some of the early systems developed by Weiser and his colleagues in more detail. Weiser defined ubicomp as follows:

> Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives.

Two parts of this definition are important. The first part suggests that computing needs to scale differently than mainframes and desktop computers. Rather than catering to one device, ubicomp assumes and designs for many interconnected devices. Rather than a system for one user, it assumes many users. Finally, rather than the desktop, it spreads into larger environments (e.g., rooms, homes, cities).

The other key part of the definition of ubicomp proposes that technology needs to recede into the background of our lives. Technology should be calm. Alternatively, researchers have talked about technology that is unremarkable [804], invisible [584], or disappearing [779]. The idea here is that computing becomes physically integrated with our surroundings in addition to becoming cognitively less prominent. Information should be delived in the right place and at the appropriate time.

Ubicomp has been enabled by advances in computing, particularly networking, miniaturization, and energy-efficient sensors and processors. Fulfilling the vision of ubicomp continues to raise challenges for research and development in computing. However, the focal point of this section is how ubiquitous computing has affected those points of computing that face users.

Four principled differences to desktop computing stand out. First, input and output become *embedded* in the environment. Second, *context-awareness*: sensor data may be used for recognizing what users or doing or in which context interaction takes place. Third, interaction become partially *implicit*, intending to draw on users' natural ways of engaging with each other and the world. In other words, users may not be needing to give explicit commands to a computer but, thanks to context-awareness, it could trigger actions on its own. Fourth, the substrates of the user interfaces change from the desktop and its well-known organization to objects, homes, classrooms, and the world. In short, computing becomes *smart*. The following subsections outline these different ways of thinking about user interfaces and summarize their benefits and drawbacks.

29.2.1. Embedded Input and Output

Ubicomp extends the types of input that is used for its user interfaces significantly over that used in desktop computers. As computing multiplies and is merged with the environment, we get an increasingly varied and fine-grained set of input about users. This input may come from sensors may be embedded in objects, from sensors distributed in the environment, or information from the devices that users carry with them.

Consider the ActiveBadge system as an example, an early ubicomp system [852]. The system was produced at a time where wireless networks did not exist when telephones were rarely portable, and when a frequent way of contacting people was to use a pager, a device which would simply make a sound and show a number, when somebody wanted to chat with you. In contrast, ActiveBadges would transmit a brief, unique infrared signal every 15 seconds. This could be picked up by a sensor-network and be used to locate an individual wearing the badge to a particular room within a building.

Many other types of input has been proposed since the 1980s. Similarly to mobile computing, this has concerned the location of devices or of the user. For instance, [505] captured the proximity of users to devices. They showed how to capture and use a person's distance to a display for modifying the contents of the display. [163] sensed electromagnetic noise using the human body as a receiving antenna to recognize gestures of the body as input.

It is not only input that may be embedded in the environment; *output devices* may also be embedded in the environment. The key idea here is to design hardware that allow information to be displayed to the user in a calm manner. Such output has been called ambient media [368] or peripheral displays [511]. Typically, there is no interaction possible with such displays, they merely present information.

An early example of using embedding output in the world was Jeremijenko's Dangling String, described by [861]. This art piece is a 2.5m piece of plastic that hangs from an electric motor mounted in a ceiling. The motor moves the plastic based on the amount of data being transmitted over the network; "A very busy network causes a madly whirling string with a characteristic noise; a quiet network causes only a small twitch every few seconds".

Another example is the AmbientRoom [368]. AmbientRoom uses light, shadow, sound, and airflow to present information in the perifery of users' attention. For instance, one may be interested in a particular product that just have been launched on a website. Each view of the product webpage may be presented as the sound of raindrops. One might notice that this soundscape changes from dry to heavy rain and wish to investigate.

29.2.2. Context Awareness

One of the main innovation for HCI developed in ubicomp research is techniques for *context awareness*. A computer system is context-aware "if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task" [187, p. 5]. Desktop computers have little sense of context but with new sensors and new modalities we can characterize users, their activities, and their tasks.

Sometimes context may be defined in a straightforward manner; the geographical location of a user might be one example. That might make it clear, for instance, whether a user is at home or at work. However, many other types of context are possible. [720] separated two types of context. *Human factors* concerns (a) information about the user, such as their emotional state or their habits; (b) the social environment of the user, such as the collocation of other people; and (c) the user's task. Information on the *physical environment* includes (d) the aforementioned location; (e) infrastructure, such as nearby resources for computation or communication; and (f) the physical conditions, such as noise and light. Each of these may be sensed and used in context-aware applications. Human factors are often hard to infer from sensor data, as they constitute latent (unobservable) mental events. Hence, interactions should be designed in such a way that the system's actions can be configured and corrected by the user.

In other cases, we need to do sophisticated analysis of input to infer context. A large portion of this work has been about *activity recognition*. Activity recognition concerns inferring the activities of people based on input data. One example of this was done by [243]. They wanted to recognize activities in the home without the cost of instrumenting all rooms in a house. They deployed low-cost microphones on the cold and warm water pipes and on the drains. From the microphones, the authors were to train classifiers that could predict which appliances where used, which parts of bathrooms that were used, and when there was something being done in the kitchen. Sometimes activity recognition concerns mental phenomena like 'being busy' or 'being stressed'. Other examples of activity recognition includes being able to detect boredom of smartphone users [638] or when smartphone users are lying [545].

Acting on context can be done in at least three ways [187, 720].

- Services may run automatically or proactively based on users' context. For example, lighting can be changed based on the user's activity at home (e.g., working vs. cooking).
- Depending on the context, different information or services may be offered to the user. The ActiveBadge system mentioned earlier [852] used context to change when and how calls were put through: "most people would prefer not to take unexpected telephone calls when they have just been called into their boss's office; others might not want to receive calls if they are in the lunch room between 12 and 1pm".

Question	Challenge	Possible problem
How do I address one or more of many possible devices?	How to disambiguate signal-to-noise? How to disambiguate intended target system? How to not address the system	No response, unwanted response
How do I know the system is ready and attending to my actions?	How to embody appropriate feedback so that the user can be aware of the system's attention; how to direct feedback to the zone of user attention	Wasted effort, unintended actions, privacy and security concerns
How do I know the system is doing or has done the right thing?	How to select objects? How to show system state? How to bind actions to objects?	Few operations possible, failure to do actions, unintended actions
How do I avoid mistakes	How to control or cancel actions? How to intervene when users make obvious errors?	Unintended actions, unintended results, inability to recover state

Table 29.1.: Sample questions for interaction with ubicomp systems; adapted from Bellotti et al. [60], p. 417.

- Context may be tagged so that it can be acted upon by the user. In the Classroom 2000 system, for example, university lectures were recorded alongside whiteboard markings [5]. That integrated capture and tagging produce a rich set of notes from the lecture. Later work has focused on so-called life logging, that is, integrated capture and tagging of everything an individual experience. In the *Forget-me-not* system, for instance, a users is shown for each day information on whom they met, which documents they printed, and which calls they made.

29.2.3. Natural and Implicit Interaction

A main point of departure for many ubicomp systems is that interaction should be *natural* in its context. Thus, much work on input using gestures, movement, and speech has been done. A related characteristic of interaction styles in ubicmp is that it becomes *implicit*. In graphical user interfaces, users give explicit commands. Much interaction in ubicomp is command-free, using sensed input. The concerns around most interaction techniques for ubicomp, then, revolve around whether users feel as the agents behind interactions and whether it is clear to them that they are indeed interaction. For instance, many people have struggled with automated lighting, waving their arms to make the light go on and wondering why the light was switched off, even if they were still in a room. [60] offered a set of questions about implicit interaction in ubicomp systems (see Table 29.1).

29.2.4. Is implicit interaction realistic?

Although ubicomp has been researched since the 1990s, it is still an active area of research. The main benefit from HCI's point of view is that computers can be used in everyday contexts as opposed to dedicated workspaces. While Weiser's tabs, pads, and boards are already a reality, realizing the stronger vision, the idea of implicit interaction, has been difficult to realize. Recognizing from sensor data what a user does or what the user wants has turned out to be very hard. Whenever prediction fails, the user must have a way to correct the system or disable the system. Moreover, all sensing systems can also be used for commercial and even adversarial purposes, like monitoring users. The user pays the price for failure, be that in terms of effort of loss of privacy. If the average benefits do not justify such costs, users may be reluctant to start using these systems. Concerns like these has led to researchers to look at *intellegibility* and *controllability* and *personalizability* of ubicomp systems. However, besides succesful case examples, no general solutions exist for these challenges.

Paper Example 29.2.1 :

The challenge of sensing complex states.

One goal of ubicomp has been sensing, also the internal states of people. The research area of affective computing has focused on sensing human affect and adapting interactive systems based on the sensed information. If you recall the chapter on emotion (see Chapter 7), you will remember that theories of human emotion are complex. Moreover, getting the sensing of some aspect of emotion (like core affect) right has required significant effort.

Picard et al. [637] explored how to decode physiological signals from a user into a prediction of an affective state for that user. The authors were particularly interested in exploring the robustness and variability in the sensing methods. To that end, they collected about a month of data from a person who tried to experience particular emotions. The person was asked to go through eight emotions in turn, including anger, hate, grief, and joy.

While people tried to experience the emotions, four measures were collected. (1) facial muscle activity, (2) blood pressure, (3) skin conductance, which is increased by sweat on parts of the hands, and (4) respiration rate. Based on these measures, the authors developed a variety of models for accurately predicting emotions, solving variation among days of the features that indicated the relevant emotions. Although affective computing is still difficult, but this study was an early demonstration that some variation in measures can be dealt with.

Note also that Picard et al. [637] used a single participant only, making the point that research in HCI need not have many participants. In this case, the strength of the data is that is spans many weeks.

29.3. Tangible User Interfaces

Desktop user interfaces, in particular the GUI, use a generic input device (often a mouse), have a single user controlling that mouse, and employ a digital representation of the content that is interacted with. During the late 1980s and early 1990s, an increasing

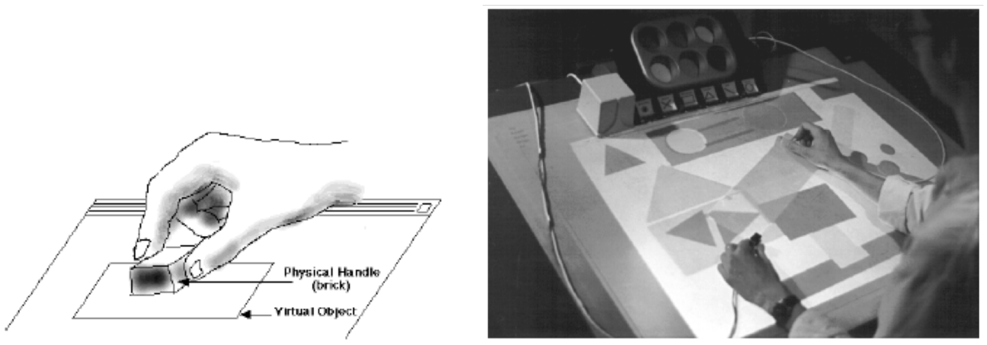


Figure 29.3.: A graspable user interface from [240]; later, the predominant way of talking about such interfaces became tangible user interfaces. The left-hand part of the figure shows how a virtual object (a window) can be selected and subsequently moved using a physical handle (called a brick). The right-hand part of the figure shows a drawing application, GraspDraw, with two active bricks connected to a computer with cables. Those bricks can are used as part of drawing, making the control of GraspDraw both more physical and allowing for two-handed interaction. The interaction happens on an inclined desk with rear-projection.

number of researchers found this form of user interface fundamentally limited. [863] expressed the limitation as follows (p. 24):

> We live in a complex world, filled with myriad objects, tools, toys, and people. Our lives are spent in diverse interaction with this environment. Yet, for the most part, our computing takes place sitting in front of, and staring at, a single glowing screen attached to an array of buttons and a mouse. Our different tasks are assigned to homogeneous overlapping windows. From the isolation of our workstations we try to interact with our surrounding environment, but the two worlds have little in common. How can we escape from the computer screen and bring these two worlds together?

Tangible user interfaces developed as an answer to this question. The overall ambition is to make interaction with computers more physical, both in input and in output. Figure 29.3 shows a classic drawing system that support input through physical manipulation and output projected on a desk [240]. Compared to desktop computing, this system transforms drawing to a physical and two-handed activity. It was one of a series of systems appearing around the same time that showed that the vision outlined in the quote above was within reach. Another well-known example was Durrell Bishop concept of a marble answering machine. It represented voice messages to a phone as marbles, physically illustrating the number of incoming messages and allowing the user to manipulate each message physically.

29.3.1. The Elements of Tangible User Interfaces

Compared to the GUI, tangible user interfaces change most components of user interfaces (e.g., devices, interaction techniques, representations, and substrates). Input devices are typically physical objects that are sensed to use their position and orientation as input. This may happen using RFID tags that are embedded in the objects, with tracking of recognizable visual markers (so-called fiducials, for instance tracked from beneath a surface), or sensed from the inside of the object (for instance, as the proximity to other objects). For instance, the reacTable allows physical objects that represents samples and filters to be placed and tracked on a tabletop display using a marker under the object [387]. Output devices have often been projectors, showing digital content amidst physical objects, or horizontal displays, on which physical objects may be placed (as in reacTable). Output may also be realized through actuation of real-world objects. [286] placed small figurines atop a servo motor; the way the figurines were facing reflected the presence or absence of remote coworkers. In Madgets, output from the computer may be given to physical widgets that can be electromagnetically moved around on a tabletop [862]. In these cases, the physical widgets enable the user to control the virtual content *and* the virtual content may also control the positioning of the widgets. In that way, input and output melt together.

A key goal of interaction techniques for tangible user interfaces is to mimic interactions with physical objects. Fishkin referred to this as metaphor of action. For instance, rotating a scan of the brain on a display may be accomplished by rotating a fist-sized model of a head [336]. The IO brush allows users to pick up visuals from their surroundings using a brush-like device with an embedded camera [701]. Users may then use those visuals to draw. Interaction techniques may also use constraints imposed by the physical world to steer interaction. In the marble answering machine, a small dent in the machine indicates and constraints where marbles may be placed. In Pico, actuated objects could be constrained in where they could move by the user's hands and by physical props, like elastic bands between objects or physical barriers [626]. Finally, some interaction techniques have been modelled over combinations of physical objects. For instance, objects may be placed on top of each other, next to each other, or assembled as an interaction. Topobo allowed users to snap together physical objects to create new forms and record movement of those forms for later playback [667]. Such constructive assembly is a widely used interaction style in TUIs that support learning. Interaction style may couple input and output more or less tightly, Fishkin called that the degree of embodiment. For instance, the model of the head previously mentioned is coupled to output on a display (distant embodiment) whereas IlluminatingClay uses clay both as input and output (full embodiment), see [642]. In the latter case, interaction techniques ensure that physical objects are simultenously acting as representations and as controls [817].

The representation of information in tangible user interfaces is shaped by it being bound to physical spaces and objects. The physical objects themselves may be generic, say, the marbles in the marble answering machine. In Fishkin's terminology, this is a question of the metaphor of the noun or the shape of the object; bricks use no metaphor and are dependent on convention or learning. Generic objects allow the user to couple physical objects to interface objects or actions; [341] called such objects containers (because they may represent any kind of information). Generic objects may be augmented or annotated with information to help distinguish them from each other. The physical objects may also be non-generic; [341] called such objects tokens. For instance, in the urban-planning system Urp, users give input by moving physical architectural models on a table surface [820]. The models and the movements of them help calculate the shadows for arbitrary times of day and output is given by projecting them back over the models. Physical objects may represent actions. The bricks in Figure 29.3 serve as generic representations of actions like move and rotate.

Perhaps the most prominent characteristic of tangible user interfaces is that the substrate of the interface is the physical world. [352] argued that "tangible interaction is embedded in real space and interaction therefore occurs by movement in space" (p. 439). Thereby, computing is moved out of the computer; the social and spatial aspects of the world thereby become key parts of the interaction. In many projects, the real-world substrate has been tabletops (e.g., Urp, graspable user interfaces as in Figure 29.3). Many visions of tangible computing suggest using the real world to embed information. As one of three essential characteristics of tangible interfaces, [368] listed the "use of ambient media such as sound, light, airflow, and water movement for background interfaces with cyberspace at the periphery of human perception" (p. 235). While this idea has been widely explored, it does not necessarily imply mean that the any part of the interface can be physically manipulated.

29.3.2. Models of Tangible User Interfaces

Figure 29.4 shows a simple model of tangible user interfaces called MCRit [817]. This model was formulated in opposition to a prominent model used in graphical user interfaces called model-view-controller (MVC; see Chapter 38). In MVC, the model is independent of the user interface and manages the data and logic of the system. The view makes information accessible to the user, in graphical user interfaces, typically in a visual form. The controller takes input from the user and uses it to affect the model or the view. Thereby, views and controls are logically separated.

In contrast, MCRpd stands for model, control, representation—physical and digital. Compared to MVC, the model plays a similar role, comprising the digital content of the system. The view component, however, is replaced by representations. Those may be physical (e.g., bricks) or digital (e.g., video, audio). In contrast to the MVC, representations are always in the physical domain. Finally, the control component of the interface is always fully physical. Physical representations serve both to reflect the state of the digital model and as mechanisms for interactive control.

29.3.3. Pros and Cons of Tangible User Interfaces

Tangible user interfaces have many benefits over other types of user interface; see [740] for an extensive review.

- Tangible user interfaces tap fundamental human skills in using our hands and

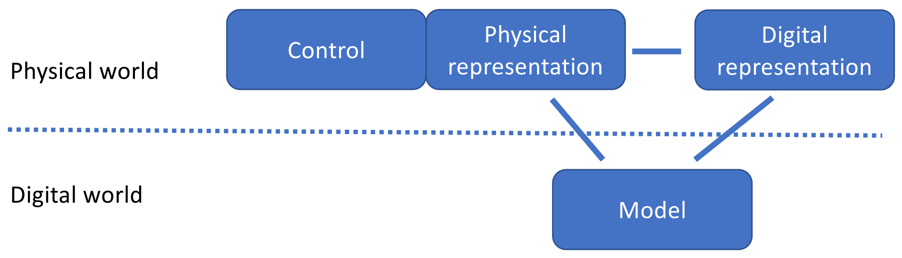


Figure 29.4.: A model of tangible user interfaces by Ullmer and Ishii [817]. The physical world reflects the digital model, both as digital representations and as physical representations. The physical representations serve as controls of the digital model.

bodies for thinking and acting. For instance, early experiments with the interface shown in Figure 29.3 showed that participants were faster with the tangible user interface, had fewer difficulties in switching between tools because the tools were physically represented, and could work in parallel relying on hand-eye coordination and proprioception. This is also evident in the many tangible projects that explore learning. Tobopo, for instance, used manual skills in assembly to facilitate interaction. In that way, tangible user interfaces tap our understanding of physics and our body awareness and skills [374], enhancing learnability and immediate usability.

- Tangible user interfaces also offer multiple devices for input. This enables the simultaneous use of multiple devices, either by two-handed interaction, or by multiple people. On a desktop computer, people would have to take turns (or time-multiplex); in tangible user interfaces, they can act simultaneously (or space-multiplex). Reactable, for instance, allows multiple users to simultaneously manipulate sound effects and samples due to its round shape and multiple tokens [387]. This supports social awareness and skills [374].
- Tangible user interfaces uses specific physical objects, rather than generic input devices. In that way, what the objects do and how to use them are more readily apparent to users. This means that actions are easily discoverable and that the gulf of execution is lessen by the specificity of the physical objects.
- Tangible user interfaces integrate control and representation. Thus, the control always reflect the digital model. Thereby, feedback is integrated and direct. In tangible user interfaces, there are rarely issues of mapping because of this direct integration.

At the same time, tangible user interfaces have limitations. Tangible user interfaces comes with the limitations of being tied to physical objects: there are just one of an object, they may be lost, and they may take up all the workspace or otherwise result in clutter.

While this sounds like a practical concern, the specificity of physical representations imposes severe restrictions on how well tangible user interfaces work for actual tasks.

Tangible user interfaces do not well support abstraction. Most operations are direct manipulation and thereby share the limitations of that style of interaction by not allowing abstract operators (e.g., wildcards) or syntax-based manipulations. This several limits their expressivety, that is, how much may be expressed in a command, as well as their scalability, that is, how many objects they maximally work with . Perhaps for this reason, ultimate performance with tangible user interfaces remains low.

A third issue is that tangible interfaces need to fully align virtual and physical models. The issue here is both to make physical models accurately and quickly reflect changes in physical models, for instance with respect to the position of an object. The issue is also related to retaining the properties of physical objects while using them (and all of their properties) to control digital models.

The development and evaluation of tangible user interfaces is an active research field and there is strong community involved in furthering the vision described in the beginning of this section, see for instance papers and prototypes published at the annual conference Tangible, Embedded, and Embodied Interaction.

29.4. Mixed Reality

Mixed reality (MR) refers to user interface technology that mixes virtual and real content to create a new, augmented experience that is interactive and appears authentic to the user. The two defining aspects of MR are (1) a programmatic association between virtual contents and real world and (2) an interactive display of that association.

Figure 29.5 shows the first virtual reality system built by Sutherland and Sproull. The system, called the Sword of Damocles, as it hang over the head of the user, was built in 1968, was inspiration for all future mixed reality applications based on head-mounted displays (HMDs). Sutherland described his vision as follows:

> Don't think of that thing as a screen, think of it as a window, a window through which one looks into a virtual world. The challenge to computer graphics is to make that virtual world look real, sound real, move and respond to interaction in real time, and even feel real.

The system developed consisted of a headband with CRTs (cathode ray tubes) used to project to two lenses. The glasses showed vector graphics such as a molecule or virtual room. The camera view was updated when the user moved. The key innovation was that the head position of the user was tracked and linked to position of camera in the virtual world. The authors argued that rapid response to the movement of the head is a more important requirement for creating an illusion of three-dimensionality than having authentic stereoscopic cues [790, p.757]:

> The image presented by the three-dimensional display must change in exactly the way that the image of a real object would change for similar motions of the user's head. Psychologists have long known that moving perspective

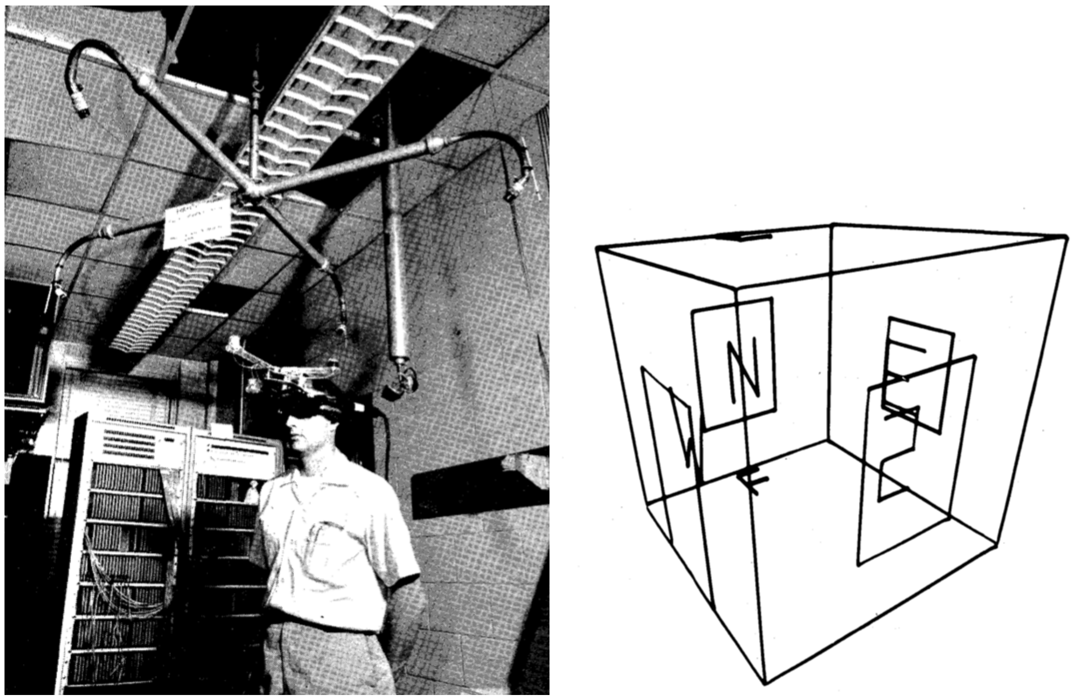


Figure 29.5.: The Sword of Damocles, the first head-mounted display developed by Sutherland and Sproull in 1968

> images appear strikingly three-dimensional even without stereo presentation; the three-dimensional display described in this paper depends heavily on this "kinetic depth effect."

(Presenting stereoscopic depth with glasses was possible only much later.) Beyond the prototype was a bigger vision of computer display as a window to a virtual world. As the virtual world was responsive and presented in an immersive way, it would feel real.

In mixed reality, the relationship between the virtual and the physical is understood as a graded one. It is not an all-or-none decision. The *mixed reality continuum* of Milgram and Kishino was an attempt to define this relationship more precisely:

> Our objective is to formulate a taxonomy of the various ways in which the "virtual" and "real" aspects of MR environments can be realised. The perceived need to do this arises out of our own experiences with this class of environments, with respect to which parallel problems of inexact terminologies and unclear conceptual boundaries appear to exist among researchers in the field.

The continuum is presented in Figure 29.6. At one extreme, there is the real environment and, at the other, virtual environment. The zone in-between them is called mixed reality, which is further divided into two sub-regions: augmented reality and augmented virtuality.

According to Milgram and Kishino, the user interface, and especially the way it couples display and the tracking of the user's motion, has a decisive role. They divided interface technology accordingly into six classes:

1. **Class 1**: Monitor-based, non-immersive displays, "windows-on-the-world". Consider for example a 'magic lens' or see-through view on a mobile device, one that overlays virtual information on a camera viewfinder.

2. **Class 2**: Video displays as in the former class, but implemented using immersive HMDs rather than monitors.

3. **Class 3:** HMD's with optically implemented see-through capability. Graphics are superimposed on see-through glasses.

4. **Class 4**: Same as Class 3 but with a video-based see-through.

5. **Class 5**: Completely graphic environments, to which video "reality" is added.

6. **Class 6**: Completely graphic environments, in which physical objects in the user's environment are embedded.

The continuum was instrumental to the formation of AR and VR as active research topics. It helped understand the design space as well as the associated engineering challenges. For example, in Classes from 2 to 4, *motion tracking* and *registration* are challenging. In The Sword of Damocles, both the head and hands were tracked. Sutherland stated it would be important that the virtual environment was not only for passive viewing but allow manipulating virtual objects directly with hands. In some of their prototypes, hand positions were tracked with fishing lines connected to reels. Although the solution appears crude, the technical problem of hand and finger tracking is still open. The issue is that any inaccuracies in respect to *world coordinates* transfer to an offset in the way the physical and the virtual environments are registered. Consider, for example, petting a dog in virtual reality with hands tracked by computer vision. If the tracker is several inches off, the hands as you feel them via proprioception will not feel like the ones that are shown on the display.

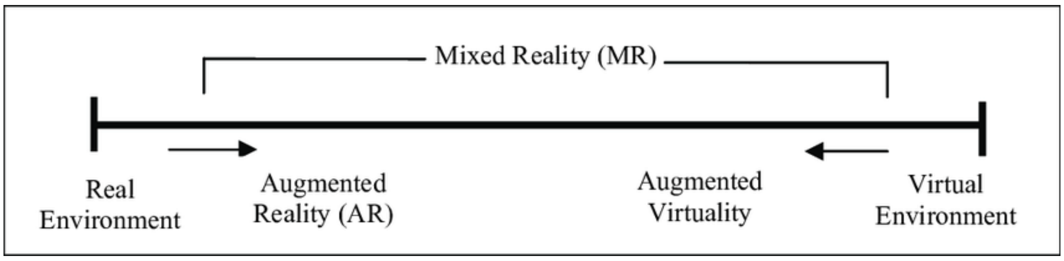


Figure 29.6.: The reality continuum by Milgram and Kishino [529].

The continuum also puts excess emphasis on visual experience [ref]. Although MR is often about visual content, content may also be delivered through sound or haptics. A virtual orchestra can be projected auditorily into a physical space. A haptics device on the finger can mimic virtual surfaces when touching them with the finger.

The interactive relationship between the real and the virtual can also be more complicated than that portrayed by the six classes. For example, *substitutional reality* refers to a relationship where an object in the physical reality is coupled to one in the real world, or 'substituted' [ref]. Selected objects are 'twins'. For example, a living room can be substituted with the command deck of a spaceship.

29.4.1. Virtual reality

Virtual reality (VR) refers to an interactive environment that responds to user's actions, for example via locomotion of camera control. 'A reality' is more than just virtual 'content'. It is usually understood as a space that binds together content. As a term, virtual reality is surprisingly new, from the 1980s, and credited with Jaron Lanier, a computer scientist and an artist who worked at Atari Inc with a team that invented the data glove. The first virtual reality system, however, even if not called by that term, is attributed to Morton Heilig. His Sensorama was a multimodal simulator. A user (or rather, viewer), would sit down and put head inside of a large box, which showed a 3D motion picture with smell, stereo sound, seat vibrations and blew wind in the hair.

Virtual reality is a multi-disciplinary topic drawing from computer science, electrical engineering, and psychology in addition to HCI. Virtual reality systems, more generally, have some sensors (to track motion), effectors (e.g., displays), and a reality simulation (e.g,. a world model in Unity). *A virtual reality system*, more precisely, is defined by seven features [Brooks]:

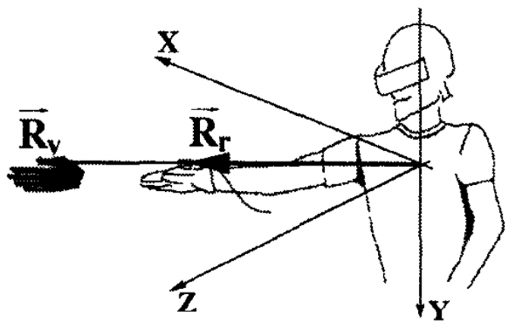
1. Blocking out sensory impressions generated by the real world (think: VR HMDs);
2. Continuously updated computer graphics creating a sense of immersion;
3. Tracking of user position and orientation;
4. Software for modeling the relationship between the virtual and the real;
5. Synthesized sound and possibly haptic sensations (think: vibrating steering wheels for car games);
6. Input devices that enable interactions with virtual objects, and;
7. Interaction techniques that substitute real interactions (e.g., hand movement)

Paper Example 29.4.1 :

Interacting with objects at a distance A problem in virtual and augmented reality is that objects can be further away than the user can reach. An early solution to tackle this problem is the *Go-Go* Go-go interaction technique [655].

The metaphor underpinning the tehnique is that the user's arm grows nonlinearly in VR when the user reaches out to touch, grasp or otherwise interact with a virtual object that is out of the user's reach. When the user interacts within a set threshold the virtual hand and the user's real hand are mapped to the same place. However, when the user extends their arm passed a threshold the virtual arm of the user, and thus the position of the user's virtual hand, starts to grow nonlinealy, allowing the user to reach virtual objects that are further away than the user can normally reach.

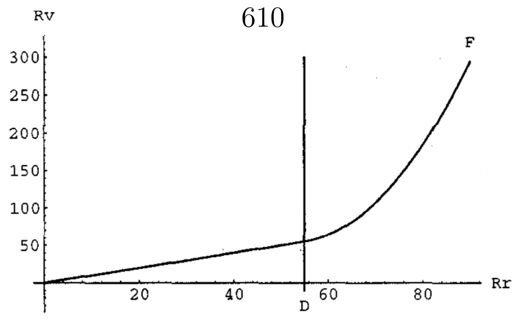
The user's real hand position, mapped by the vector $\vec{R_r}$ in the diagram below is mapped to a virtual hand position parameterized by $\vec{R_v}$. R_r is thus the length of user's real arm, which is also the length of the vector $\vec{R_r}$. The vector $\vec{R_r}$ vector points from the origin of the user's hand. R_v is the length of the user's virtual arm, which is also the length of the vector $\vec{R_v}$.



When the user moves their hand the virtual hand is mapped differently depending on where the user's real hand is located in space. This mapping is governed by computing the length of the user's virtual arm (R_v) as a function F of the user's real arm (R_r), as described in Equation 29.1:

$$R_v = F(R_r) = \begin{cases} R_r & \text{if } R_r < D \\ R_r + k(R_r - D)^2 & \text{otherwise} \end{cases} \tag{29.1}$$

There are two parameters in Equation 29.1. The first is k, which is a coefficient between zero and unity controlling the nonlinear rate of expansion of the virtual arm. The second parameter is D, which controls the length the user's real arm has to be expanded to for the virtual arm to start expanding nonlinearly. This means the parameter D is a threshold. In the original paper D is set to 2/3 of the user's real arm length. After the user extends the arm further the arm starts to grow nonlinearly. This mapping between the placement of the virtual hand and the user's real hand is illustrated in the cure below. Note that when R_r is below D the mapping between R_r and R_v is linear but when the R_r exceeds D the mapping smoothly transitions from a linear to a nonlinear mapping.



Like for all interaction techniques, HCI research aims to innovate and study techniques that are performative, intuitive, and ergonomic. However, virtual and augmented reality pose an interesting opportunity: augmenting human capabilities to operate with objects in a way that might not be physically possible. Consider, for example, flying: mixed reality makes it possible to transcend the limitations of the physical body. The related research topic is *camera control* and *control of locomotion.* Several techniques have been proposed that take inspiration from the real-world. In *egocentric motion*, the user's avatar is controlled by moving it in egocentric directions. In *warping*, the user moves by warping the body to a location by touching the destination. In *point-of-interest* (POI) control, the user selects an object (e.g., by tapping it), and the camera moves to the object.

Another challenge concerns manipulating and touching virtual objects that are beyond the physical hand reach. For example, the *Go-Go technique* uses a dynamic control-to-display gain approach (see Section X). It applies a linear or non-linear mapping that effectively warps the hand to a distant location. Moving the physical hand extends it to different locations depending on how far it is from the physical body. However, it is an open problem how to manipulate distant objects efficiently. The issue is that the further the object is from the viewpoint, the less perceptual information there is about it. Hence, manipulation of distal objects requires some way to couple camera control to interaction.

Applied psychology (esp. media psychology) has taken interest in virtual reality, proposing scientific constructs to measure how 'real' it feels. *Presence* refers to the feeling of 'being there' in the virtual world that is being portrayed, as opposed to being present in the physical, real world. The formation of this experience is understood as consisting of two steps. First, the virtual space must be mentally construed from perceivable cues, similarly as physical realities are construed. Second, the user must suspend disbelief – disbelief about *not* being there in the virtual space – and experience being located within that virtual space.

Presence as a construct has become widely used in studies of VR systems, mostly as an index of how 'good' the VR interface is. But how to measure presence? The obvious idea is to let users self-report presence. However, such reports can be 'colored' (biased) by confounds, such as the novelty effect or trying to please the experimenter. Therefore, self-reports should be accompanied by behavioral or physiological measures. For example, body convection can be used. Have you ever felt like actually falling in a VR game when coming to the edge of a cliff? Leaning back can be used as an index of how believable the VR world is.

A related research question for psychological research is that of *simulator sickness*, or the feeling of nausea during the use of virtual reality applications. Simulator sickness is more prevalent among users who report being female [ref]. According to one theory, sickness is caused by a mismatch between expected sensations and felt sensations [ref]. Consider for example sitting in a virtual car when it accelerates. Your brain might expect you to force in your body due to acceleration, change in pressure in your back, and various interoceptive sensations (feeling acceleration 'in your guts'). However, the actual sensations would be mismatched or missing. According to another theory, egocentric motion in virtual reality is particularly nauseating, because the vertigo (change of visual stimuli as it comes closer to you) is not changing as expected. It is an open problem how to alleviate simulator sickness to levels acceptable more universally.

29.4.2. Augmented reality

Augmented reality adds real-world content to a virtual environment (augmented virtuality), or virtual content to a real-world experience. Technically this requires some tracking of the user's position in the world and registration of virtual objects in the real-world coordinates. An AR system, then, requires sensors that can help position (align) the user in the physical world. In addition, it requires computations to overlay (or augment) virtual information on a view to the physical world. The result, the augmented view can be presented via different media, such as HMDs, projection mapping (using a video projector to project graphics on real-world objects), handheld devices, and even contact lenses.

Creating a perception of touching an object is an open problem for AR system. AR interactions require inferring user's intention: for example, which object does the user point at or touch? Moreover, a corresponding sensation must be created, for example via haptic stimulation. Hand-worn systems, such as gloves and exoskeletons are considered to be unhygienic and restricting. External systems like air- or sound-based stimulation are physically too weak to generate compelling sensations.

29.4.3. Pros and Cons of Mixed Reality

The promise of virtual reality is compelling and as timely as ever. It promises the possibility of being 'virtually there' []. It is already substituting our real-world experiences of leisure and business. Mixed reality, on the other hand, promises to virtuality to the real world around us, providing us capabilities that are physically not possible.

MR has been developed over the past 50 years, with periods of optimism due to novel hardware that has made MR widely available, and periods of pessimism about the fundamental limitations about this form of interaction. The pros and cons are discussed extensively in the literature. These reviews are particularly optimistic for the following reasons:

- MR revolutionarizes access to information. It embeds information in the real world where it is needed, in a useful form, just when it is needed.
- MR augments people. It allows us to do impossible things, like seeing through objects, breaking the limits of physical distance, and creating new ways of social interaction that are not limited to physical space.
- MR enables new applications in education, medicine, and science to practice, train, and learn.
- MR changes perceptions: It allows taking new perspective to the world and learn about our environments and selfs.

Are we far away from realizing these dreams? At the same time, mixed reality faces a number of fundamental limitations. Current motion tracking technology is limited. One either has to accept a highly instrumented environment or inaccuracies that hamper experience. Any offset, spatial or temporal, between the user's motion and the virtual character's motion, will degrade performance and experience, and may contribute to simulator sickness. Such offsets can also be caused by limitations of computer networks, the performance of which affect the temporal discrepancy between the physical and the virtual.

For HCI research, there is a need to develop interaction techniques that are performative in 3D interactions, learnable, and ergonomic. Users get tired for lifting their arms a lot, and may in the worst case get fatigued in the matter of a few minutes of use.

29.5. Should we imitate reality or go beyond it?

In a classic paper, Hollan and Stornetta [340] raised an important question about communication systems. They were concerned that most work on such systems were based on a "belief in the efficacy of imitating face-to-face communication" (p. 199). This belief could, for instance, be based on the idea that the richer the reproduction of the cues in proximate communication, the better the communication and the better the user interface (see media richness theory, see Chapter 9).

The research surveyed in this chapter is to some extent based on the idea that we should imitate what people do without the involvement of interactive systems. For instance, the intent of reality-based computing to draw on environment awareness and skills has led to strong systems. Likewise, the idea of natural interaction is also based on the idea that we should draw on what people find intuitive and immediate in designing user interfaces.

Hollan and Stornetta [340] pondered the limitations of this approach. In particular, they discussed whether communication technologies could ever bring us close to the feeling of being physically together. In doing so, they drew an analogy, which was intended to show a way to think differently about the design of user interfaces.

> It is customary for a person with a broken leg to use crutches, but how odd it would be if they continued to use the crutches after their leg was restored to its natural condition. In contrast, one wears shoes because they provide certain advantages over our natural barefoot condition. Special purpose shoes, such as running shoes, are designed to enhance our best performance. Now crutches and shoes are both tools of a sort, but there is a difference. The crutch is designed specifically to make the best of a bad situation -to let someone hobble around until they are back in shape. On the other hand, shoes are to correct some of the problems of our natural condition, and, in the case of athletic shoes, to enhance our performance.

In that way, enhancing, compensating, and augmenting would be alternative ways to think about the role of user interfaces in relation to their users.

Much research has attempted to devise and implement user interfaces that do not imitate the real, but attempt to move beyond it. Willett et al. [873] described how to use the idea of superpowers from science fiction to improve our perception, the bandwidth and acuity of our attention, and our ability to predict things. The whole idea is to leap beyond the natural abilities of people through interactive systems; mimicking reality is not the intention. Won et al. [892] allowed participants to control a third arm in virtual reality by rotating their arm; participants learned to control the third arm effectively within five minutes.

In sum, mimicking reality is useful for much technology development, also in HCI. Reality-based interaction provides a strong framework for thinking about such interactions. However, as a general strategy, the approach of mimicking reality is limited. The argument of Hollan and Stornetta [340] is a strong reminder why.

Summary

- In reality-based interaction, users' physical surroundings contribute to interaction either explicitly (by being represented in the interface) or implicitly (by affecting it via other factors). While mimicking reality is a great starting point, it also has limitations.

- In mobile interaction, physical surroundings not only cause pertubations and noise, but engage users' to multitask, which limits their attention to the device itself.

- In ubiquitous computing, sensor data is used to infer a user's context or goal, with the goal of better embedding computers into everyday situations.

- In tangible interfaces, computations are associated to physical manipulations of physical objects.

- In mixed reality interfaces, virtual content becomes associated with physical reality, or vice versa, allowing 'mixing' of content in novel ways.

Exercises

1. Understanding sensing systems. Pick a sensing system you use in everyday life. Use the questions by Belotti et al. to diagnose potential issues. Figure out how to improve the system.

2. Ubicomp. Take a room. Count the devices in it. Chart their interrelations. Has Weiser's dream come true?

3. Natural interaction. Is mobile interaction 'natural', how about tangible interaction? What makes something 'natural' to use?

4. Tangible interfaces. Tangible interfaces were touted as a breakthrough similar to direct manipulation and GUIs. However, tangible UIs have not found their way into everyday computing. Why?

5. Are you actually mobile when you use your mobile phone? Which aspects of mobility are on or off?