HASSELT UNIVERSITY

Abstract

School of Information Technology
Computer Science

Master in Human Computer Interaction

Designing Intelligible Visualizations To Accommodate Mid-Air Gesture Input

by Pieter De Decker

Compared to mouse, keyboard and touch screen interaction, mid-air input is a lot less established. Over the last few decades, countless techniques have been developed and integrated into UIs to teach users how to operate them. Although some existing ideas can be transferred over to mid-air interfaces, they are faced with new design challenges not found in other types of input.

In its first part, this thesis surveys a broad range of existing techniques that were developed to improve the learnability of user interfaces. The techniques are divided into 13 groups based on their (dis-)similarities. Following our study of existing techniques, we describe how we developed a visualizer that assists users during the process of executing stroke-based 3D mid-air gestures. The visualization can be integrated into any C# application that runs on Windows Presentation Foundation and uses a Kinect camera to gather user input. We organized a user test with 15 participants to see how well our visualization could guide them through the gesture execution process. 94% of them successfully executed a full set of 18 unimanual and bimanual gestures.

Acknowledgements

I would like to thank dr. Davy Vanacken, Jo Vermeulen, prof. dr. Kris Luyten, Arno Barzan, Kashyap Todi, Gustavo Rovelo Ruiz, Donald Degraen and everyone else who provided me with feedback, advice or ideas. Facilities and equipment were provided by the Expertise Centre for Digital Media and Hasselt University.

My mid-air gesture visualizer uses icons that were provided by various designers and organizations under generous CC BY 3.0 or public domain licenses.

- Hand by Megan Strickland
- Hand by Naomi Atkinson from the Noun Project
- Happy by Megan Sheehan from the Noun Project
- Kinect by Ammar Ceker
- Lock by DesignModo
- Pedal by Daniem MacDonald
- Refresh by Veille OwnWeb from the Noun Project
- Volume by Dmitry Baranovskiy from the Noun Project
- Zoom by Lemon Liu from the Noun Project
- Zoom Out by Lemon Liu from the Noun Project

Contents

Chapter 1

Introduction

Mouse, keyboard and touch screen interaction play a big role in many people's lives. Part of their success can be attributed to major usability improvements to technology in recent decades. Not too long ago, people had to learn MS-DOS commands by heart and keep a thick manual next to their computer at all times. As hardware capabilities increased, major technological limitations constraining UIs to primitive command-line interfaces were lifted. Graphical user interfaces were invented and gradually improved over the years. GUI toolkits decreased the complexity of creating and maintaining user interfaces. These new developments made it easier for software engineers to implement subtle mechanisms that were designed to assist new users, diminishing the need for paper manuals. In Chapter 2, we survey existing UI learning techniques by pulling concrete examples from software, games and academic literature. We organize the techniques that we found into 13 groups and point out their similarities and differences.

After analyzing how existing UI learning techniques are already helping users find their way within a user interface, we focus our attention specifically on gestural interfaces. Gestures are not a recent invention. In fact, the history of the pinch-to-zoom gesture can be traced back all the way to a 1985 publication by Krueger et al. [45]. Although gestural interfaces have been around for a while, they face some issues that are hard to solve. With traditional mouse input, users can see on their screen which functions are available to them. When they see a button labeled 'Save file', they can click it to save their file. They do not have to remember any vague keyboard commands like in the pre-GUI era. But how are users supposed to discover which gestures are available to them at any given moment and how they can perform them?

A lot of work has already been done in this area, some of which is included in our survey of UI learning techniques. Despite this work, the discoverability of gestural interfaces remains an issue. Norman and Nielsen have pointed out that current technology often

implements various gestures (e.g. swiping between different views), but does not always fully disclose which gestures users can do, when they can perform them and how they can perform them [63]. Gestural interfaces are easily plagued by this discoverability problem and this can significantly undermine their usability. Therefore we need to come up with good ways to communicate their capabilities.

Within the field of gestural interfaces, several subcategories exist. The concept of gestures is very fluid and can be implemented using a broad range of input devices. Users can draw gestures with a mouse or on a pen-based interface. Touch screens allow users to draw gestures on a screen's surface with their own hands. Controllers with a built-in accelerometer such as Nintendo's Wii Remote can detect the device's orientation to implement a 'tennis racket swing' gesture. Depth cameras such as the Microsoft Kinect can track the location of certain joints in the human body and use this information to process full-body gestures. Mouse, pen and touch screen gestures are constrained to 2D and have a clear start/stop point. Several systems have already been published that help users to discover, learn and execute 2D gestures. However, we would like to focus on gestures that are performed in mid-air. As we will discuss in Chapter 3, mid-air gesture interaction is more complex than 2D surface interaction for various reasons. One important reason is that the leap from 2D to 3D introduces new issues that have not yet been addressed by 2D gesture guides.

This thesis poses the following research question: how can a visualization guide users through the process of performing mid-air gestures? To answer it, we contribute a gesture learning system that helps them execute stroke-based 3D mid-air gestures with one or two hands. Chapter 3 covers the details of our implementation and in Chapter 4, we perform a feasibility study by testing the system with users. Screenshots on the next couple of pages show what the visualization looked like during our user tests.

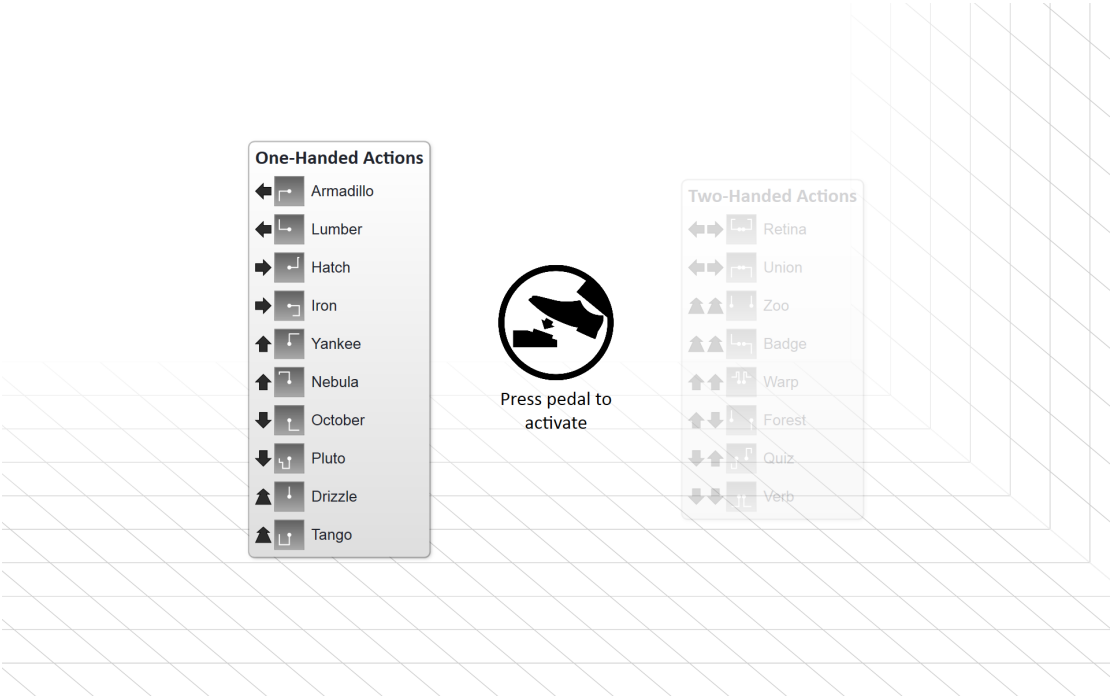


FIGURE 1.1: Pre-activation cue, shown when user raises one hand

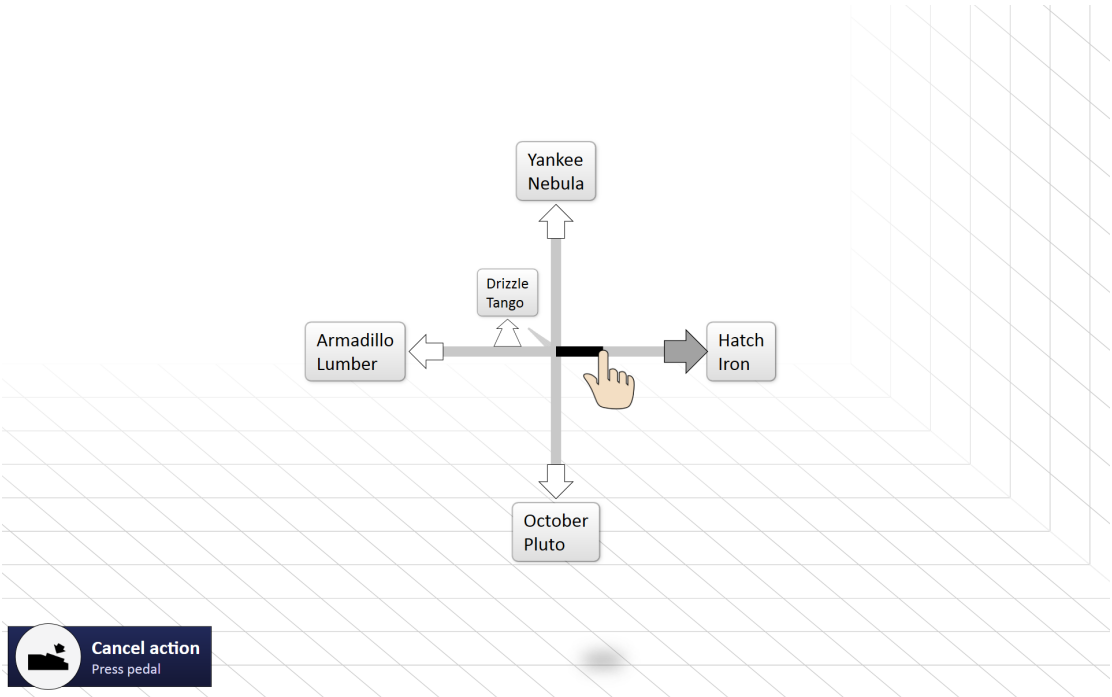


FIGURE 1.2: User starts performing a unimanual gesture

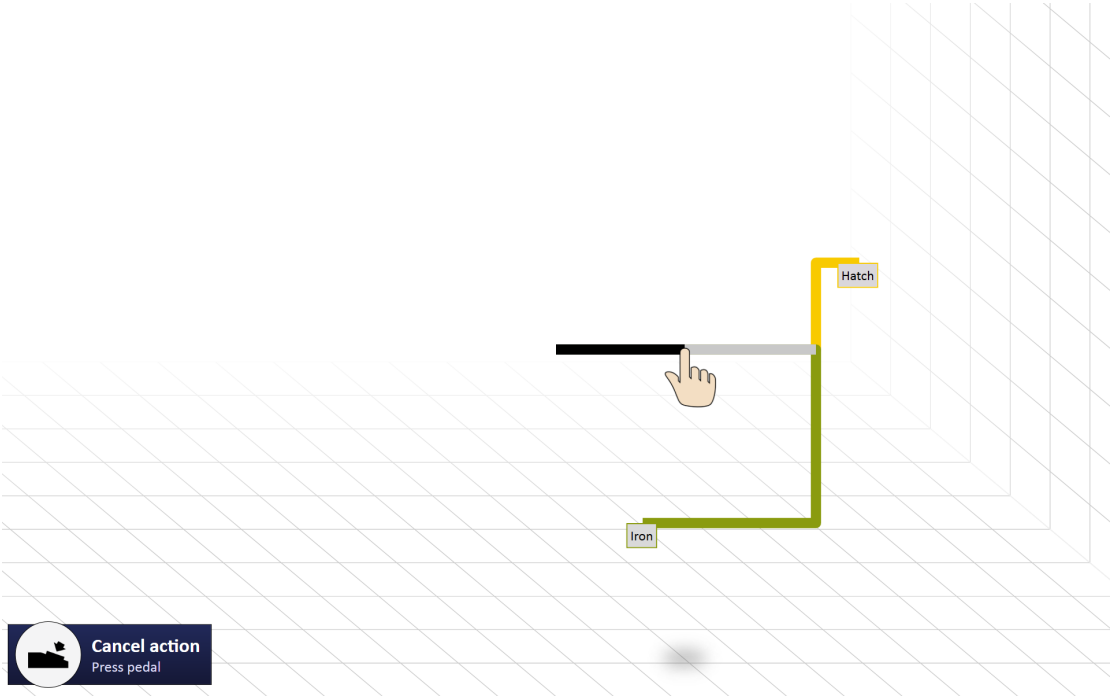


FIGURE 1.3: User will see full gesture paths when they pass the arrow

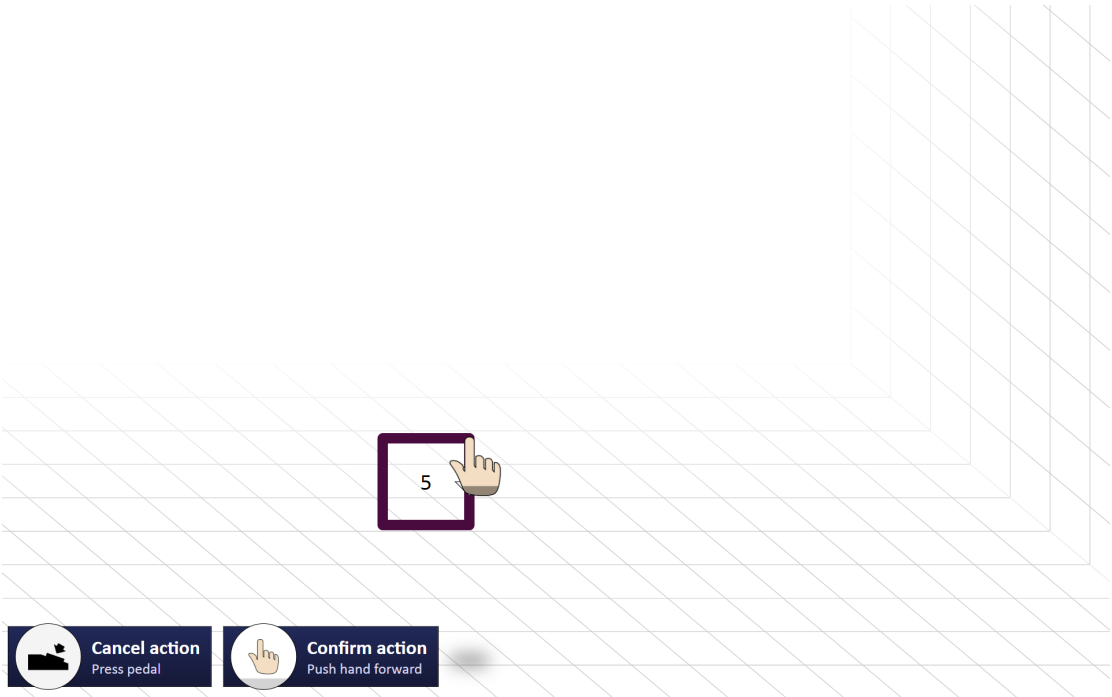


FIGURE 1.4: User performs a unimanual looping gesture

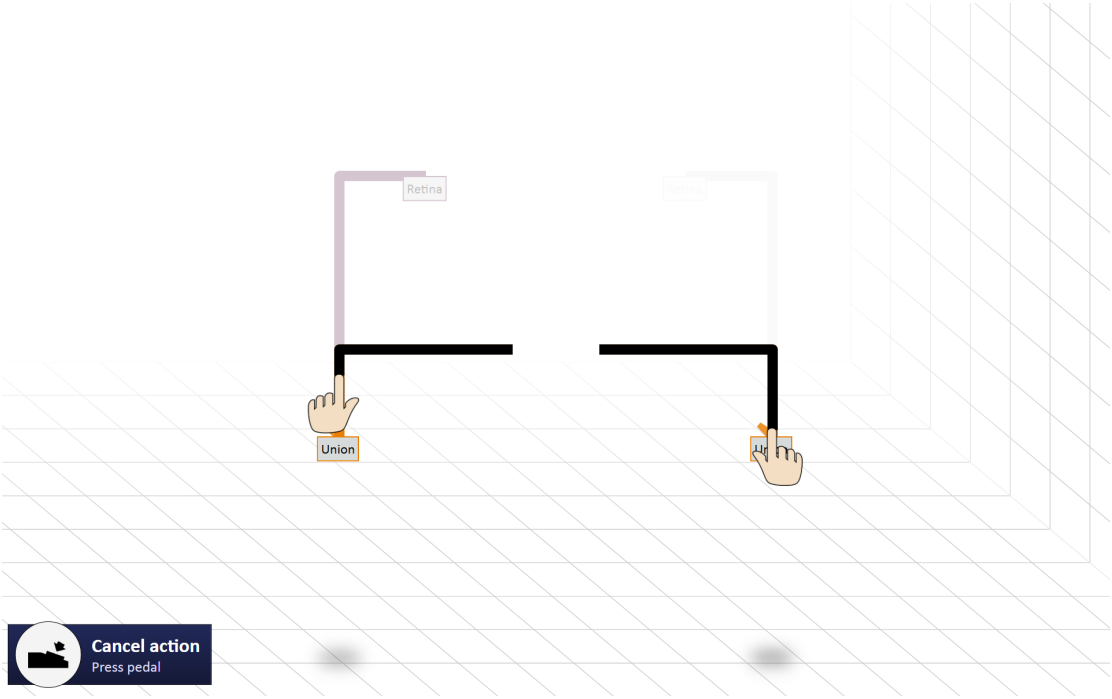


FIGURE 1.5: User performs a bimanual gesture

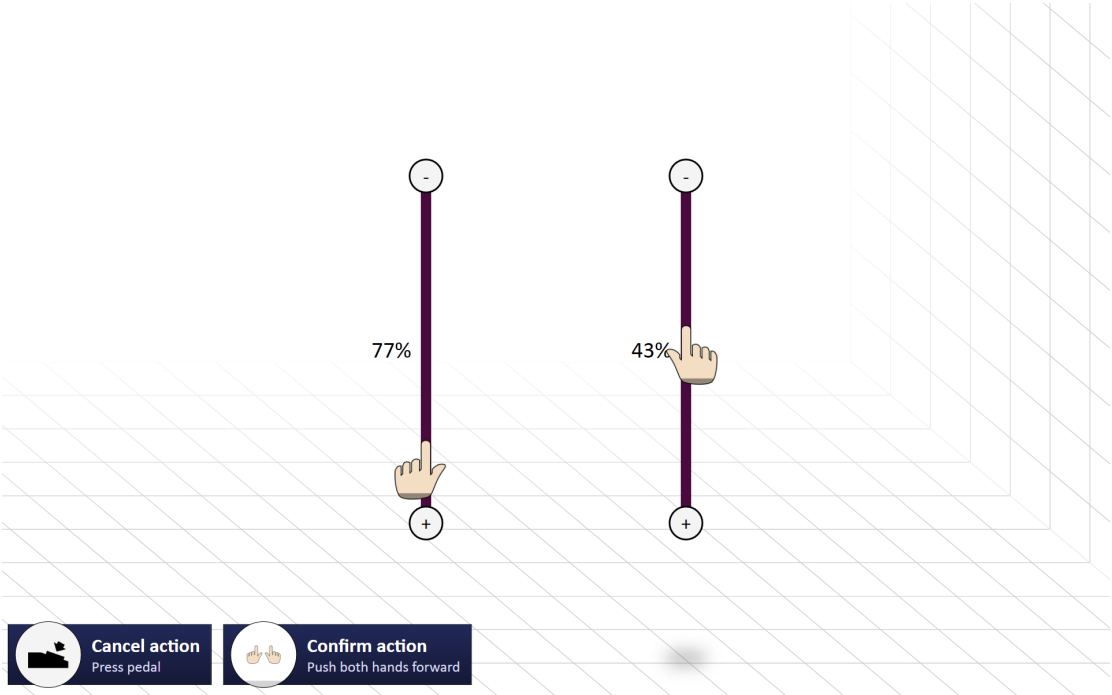


FIGURE 1.6: User performs a bimanual sliding gesture

Chapter 2

Overview and Comparison of Existing UI Learning Techniques

Humans often have a tough time communicating with machines. Sometimes we think that we are giving machines clear commands until suddenly they do something unexpected. Sometimes we spend a lot of time to figure out how we can get machines to perform a certain task for us, which ultimately leads to frustration. Usually, it is not a good sign when users have to go through great efforts to find out how they can execute a certain action. Such programs often deal with a large gulf of execution [62]. There is a middleman that has been helping us to understand machines for a long time: the paper manual. Surely it was better than pushing buttons and hoping for the best, but it was far from a perfect solution. When we encounter new technology that can speed up our work, we want to start using it immediately without spending a lot of time on training [11]. Postponing our work to thoroughly dissect a long and formal manual might be the last thing we want to do.

As primitive machines evolved into computers with rich communicative capabilities, developers and academics came up with new ways to help humans understand their mysterious and increasingly powerful workmates. In this chapter, we explore various techniques that developers and researchers have created to help people better understand everyday technology.

2.1 What is a UI learning technique?

When you are hovering over the text highlighter icon in Microsoft Word's toolbar, they provide you with a short description of its functionality. They are embodied in your

favorite role-playing game where a fictional character teaches you how to fish for food. When you select the bucket tool in Paint.NET, they tell you that you can also right-click an area to fill it with the secondary color. UI learning techniques are features that help users to understand the capabilities of software and how they can use it to achieve their goals. In terms of Nielsen's quality attributes for usability [61], they primarily show potential to improve learnability and memorability. They are reminiscent of interaction design patterns like those described by Van Welie [81], but all have the very targeted goal of familiarizing users with the UI.

Although these techniques can take on the form of a more traditional help system that is separate from the application (*Section 2.2.1*), many software and game developers have explored other avenues to provide more effective help. They will often be context-aware to a certain degree. For example, when a user clicks the 'Help' button in a window called 'Font Properties', it is not unlikely that they want to know how to create subscript text. They probably have no interest in learning how to export their document to a PDF at that point. Context-aware UI learning techniques ideally try to provide relevant assistance at the right time and in the right place, demanding as little explicit input from the user as possible.

How do UI learning techniques relate to usability theory, principles and guidelines? The techniques described here are situated on a higher level than, for instance, Norman's design principles [62]. While concepts like affordances, mappings and constraints are rather abstract, UI learning techniques are almost ready-to-use templates that leave some room for experimentation. Because they are still quite flexible, most techniques come in multiple variations and have evolved over time. Unlike Norman's design principles, they do not necessarily represent best practices, but some of them are quite effective if implemented correctly.

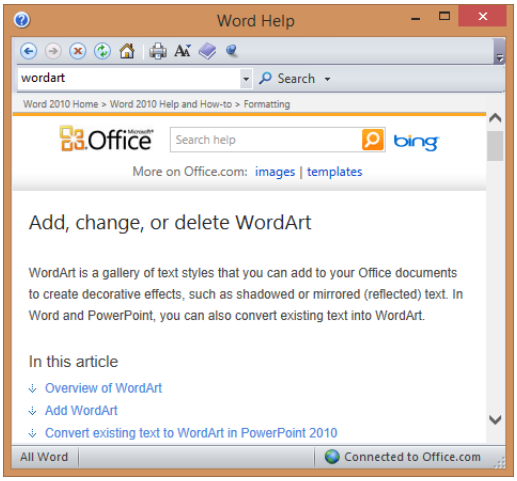
2.2 Overview of techniques

We collected existing UI learning techniques from readily available software and games as well as academic literature. This chapter provides a description for each of these techniques and cites real-world examples. We also reflect upon their usefulness, drawing from research where available. Most of the consumer products mentioned here were created for Windows, iOS or Android. While we made efforts to identify a broad range of techniques and trace them back to their earliest use, we do not claim to have a full view of what is out there.

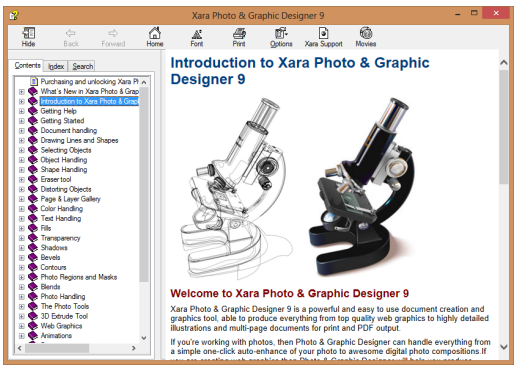
2.2.1 Standalone documentation

This UI learning technique grew historically from paper documentation and is the most traditional form of help. Standalone documentation may feature digital enhancements such as hyperlinks, collapsible sections, a search engine and embedded videos. Unlike their paper counterparts, they can use images more liberally due to the absence of printing costs. In some cases, a limited form of contextual help is provided. Such help can be implemented by allowing users to press F1 or click a 'Help' button to jump to a relevant help article. Standalone documentation can choose to retain the linear and hierarchical structure of paper manuals, but not all implementations do. In the absence of such a structure, standalone documentation can be accessed through a search engine, by invoking contextual help or by consulting an index. Either way, almost every system that we are familiar with presents help in the form of individual articles.

Microsoft Office Help Viewer [64] (*Figure 2.1a*) and Google Chrome's online help [14] are some examples of standalone documentation in popular software. Software developers can adopt a generic documentation framework such as Microsoft's HTML Help [54] to save time creating traditional help.



(A) Word 2010
Microsoft Office Help Viewer



(B) Xara Photo & Graphic Designer 9
Microsoft HTML Help

In *The Paradox of the Active User*, Carroll et al. [11] describe a phenomenon known as production bias. Users are primarily focused on getting their work done. They are, however, not motivated to learn a lot about the system or to figure out how they can work more efficiently in the long term.

> "Learners at every level of experience try to avoid reading. In structured practice, they often accidentally or deliberately get off track and end up in

> lengthy and complex tangles of error recovery or self-initiated exploration."
> – Cited from *The Paradox of the Active User* [11]

Work by Grayling [28] further reinforces the belief that traditional learning techniques are missing their impact. The author reported the following observations:

> - Users ignored the Help menu and used it only when truly desperate (...)
> - When they did reluctantly go to the help, they read the topics hastily and inaccurately. They were careless in choosing hyperlinks.
> - They bailed out of the help system early, even without finding the needed information.
>
> – Cited from *If We Build It, Will They Come?* [28]

The authors concluded from their research that users dislike conventional help and suggested that more attention should be spent developing better user interfaces, which would ideally decrease a user's reliance on standalone documentation. Embedded help was mentioned as a likely improvement over standalone documentation, but they emphasized the need for more published testing. Further discussion of embedded help can be found in Section 2.2.2.

Matejka et al. [53] have made an effort to lower the threshold to standalone documentation by creating **Ambient Help**. As shown in Figure 2.2, it automatically presents contextually relevant help articles on a second screen. In addition, multiple YouTube videos are fetched that may be relevant to the current task. The authors took the following measures to make sure that the system does not get in the way of the user's work flow:

- The audio is muted and the videos are dimmed while the cursor is in the main application. When the cursor is hovering over a video, it is played back at full brightness.
- The frame rate of the videos is reduced so that a new frame is only shown every 5 seconds, with a smooth 1-second fade between frames.
- Users can "check" a video so that it keeps playing undimmed at a low volume while their cursor is in the main application. This also prevents the video from being replaced before it finishes playing.

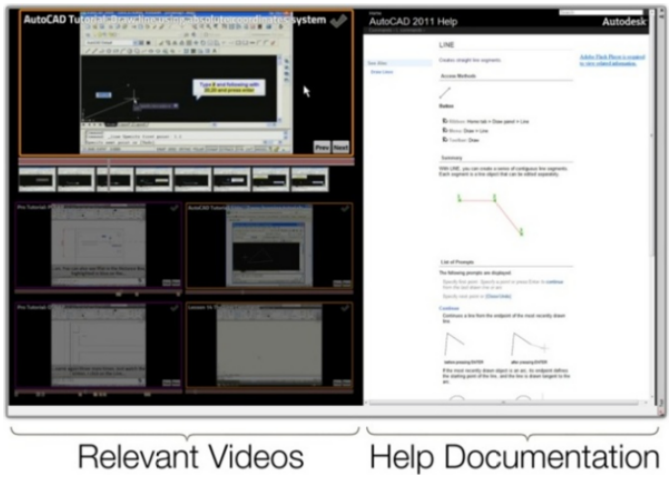


FIGURE 2.2: Ambient Help's context-aware second-screen help interface (source: [53]).

The researchers paid 12 AutoCAD users to volunteer for tests that evaluated whether Ambient Help could be useful in work-like situations. They also wanted to compare distraction and productivity levels with normal help. A study was designed to compare the performance of Ambient Help to AutoCAD's manual during intense and casual tasks. The authors concluded that Ambient Help was generating 2.6 times as much useful insights during casual tasks as the control condition. Usefulness was measured by asking questions to participants every time they looked at the secondary monitor for more than 10 seconds. They did not find any evidence that the help system hindered performance during intense tasks. It was suggested that future work should investigate how the system performed in the long term and in the user's own environment.

2.2.2 Embedded help

If the user does not want to come to the help function, perhaps the help function should come to the user. Embedded help is trying to do this in a fairly literal way. In its simplest form, help articles are embedded into the UI of the application like in Figure 2.3. Such a move lowers the barrier to help because few or no explicit interactions are needed to bring it up. Embedded help can be very context-aware. It can cater its contents to what is currently shown on screen and can take the current application state into account. Grayling [28] maintains a very broad definition on embedded help where - as we understand it - any form of help that is integrated into the application's UI fits their definition. This includes tooltips and what they referred to as 'drill-down links'. In our taxonomy, both of those examples fit into other categories of techniques. 'Text in the user interface' and 'Embedded help panes' do fit into our vision of embedded help since they more closely resemble what users might expect from standalone documentation.

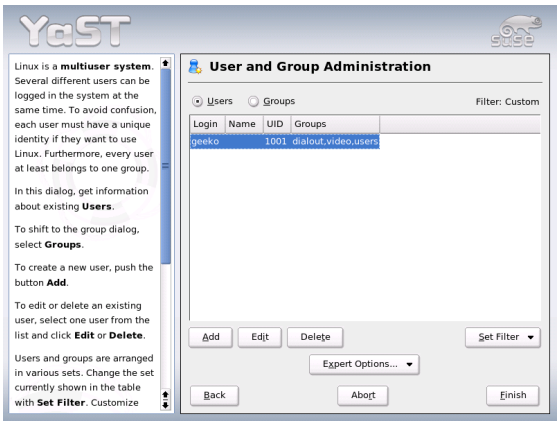


FIGURE 2.3: SuSE Linux 9.2 (2004) embeds help articles into control panel applets (source: [77])

We have found one instance of embedded help being used in message boxes. Microsoft Word 2010 [64] will display a warning message if the user is about to adjust the page settings in a way that possibly makes the document unprintable. To help confused users, the dialog shown in Figure 2.4 also contains a 'Show Help' button that reveals embedded help within the message box.

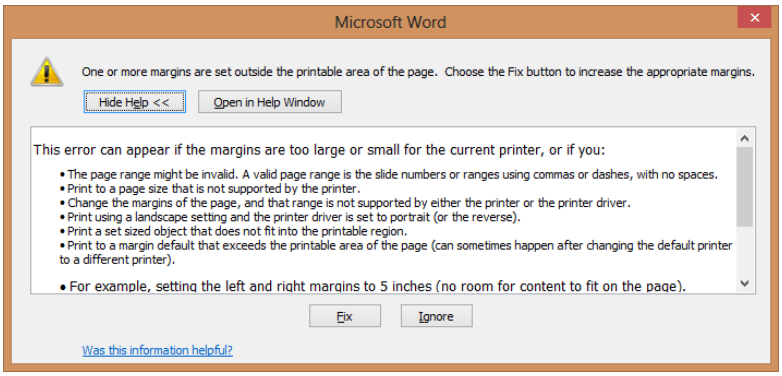


FIGURE 2.4: Embedded help in a message box from Word 2010

QuickBooks 2005 [37] had a help feature that was advertised as follow-me help [71]. This embedded help pane implements contextual awareness by adjusting its contents based on what the user is doing. For example, when users open an Invoice form, the pane will display help on invoice creation. The "How Do I..." section of the pane provides additional help topics that may be relevant at that time. One of the problems we anticipate is that follow-me help could make users feel self-conscious and uncomfortable, since a know-it-all algorithm is reacting to everything they do in a highly visible manner. Although the system was designed to help users, they may end up feeling like a robot is judging their

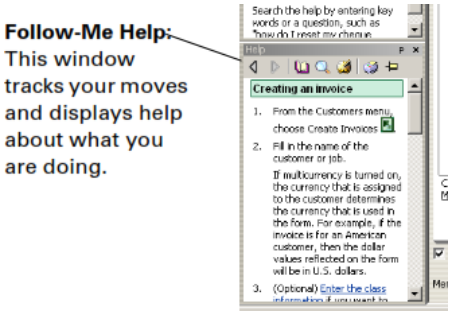


FIGURE 2.5: QuickBooks 2005's student manual explains their embedded help pane (source: [71])

perceived incompetence. This remark may also apply to certain embodied agents like Clippy (*Section 2.2.5*). We were unable to find any mention of follow-me help in the most recent version of QuickBooks, which was 2013 at the time of writing.

Not all embedded help is necessarily context-aware. Microsoft Office 2003 implemented a sidebar that displayed some generic help links as well as a search box.

2.2.3 "What's this?" help

Pictured in Figure 2.6, this form of contextual help has been around since at least the 1990s. Windows 95 and Windows NT 3.51 implemented a help mechanism that explained features and UI controls [33]. Interactions consist of two steps:

1. The user **clicks** the **question mark** button in the top right corner of the window.
2. The user **clicks** the **widget** they would like help for.

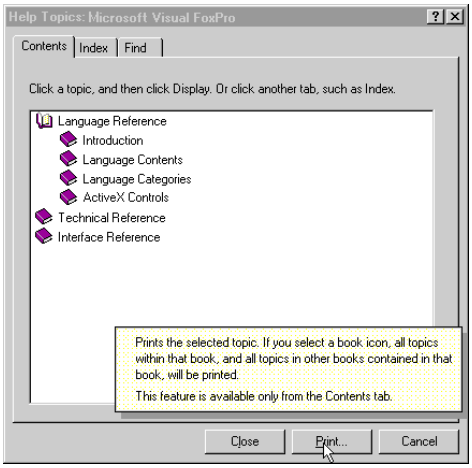


FIGURE 2.6: A "What's this?" popup in Visual FoxPro (source: [35]))

Although a 2012 article from MSDN still references this Windows feature [17], the original incarnation of this technique has become a rare sight in present-day Windows applications. Older versions of Word used to incorporate "What's this?" help [56], but later versions adopted tooltips instead (*Section 2.2.4*). This form of help only requires a hover event as user input and does not wait for the user to actively request the information.

We spotted additional "What's this?" variations on websites. YouTube Analytics (*Figure 2.7*) often displays question marks next to terms or UI controls. When clicked, additional help is revealed in a popup. This method eliminates the two-step procedure required for traditional "What's this?" help, but does not really fit into the tooltip category. Instead of relying on hover events, these question marks can only be triggered using explicit mouse clicks.

An auditory variation on "What's this?" help can be found in video games. Certain point-and-click adventure games let players right-click objects to hear a description. The 1997 title *Spy Fox in "Dry Cereal"* implements this mechanism. We also found audible versions of this technique on iOS and Android. Both platforms provide a screen reading service to assist visually impaired users: VoiceOver [85] for iOS and TalkBack [78] for

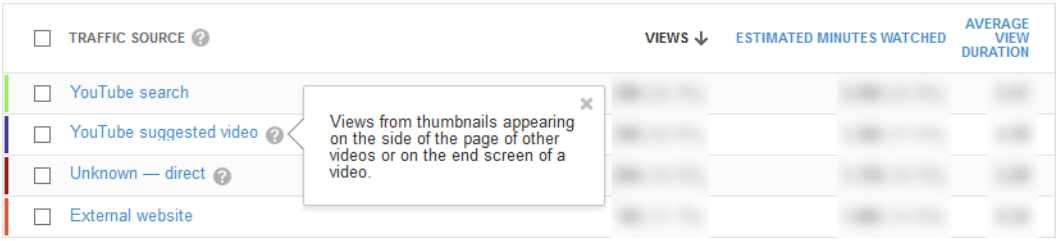


FIGURE 2.7: Clickable question mark icons in YouTube Analytics

Android. TalkBack remaps the one-finger tap as the help triggering event and provides a spoken description of elements when they are touched. For example, when a user taps the category list inside the Settings app, TalkBack says *"List showing 15 items: Location access, Security, Language & Input, (...)"* When tapping an on/off slider that is turned to the 'on' position, TalkBack says *"On checkbox, checked"*. Although this type of "What's this? " help provides UI descriptions rather than functionality descriptions, it still fits the category.

2.2.4 Tooltips

Tooltips are close relatives of "What's this?" help (*Section 2.2.3*) that do not depend on conscious user interaction. Whereas the latter requires that users (1) explicitly activate the help feature and (2) click the widget, tooltips rely on hover events.

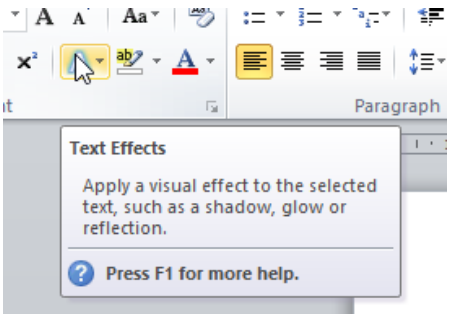


FIGURE 2.8: A tooltip from Word 2010

As is the case with "What's this?" help, tooltips commonly use a widget-centric approach. Unlike "What's this?" help, tooltips provide contextual help before users consciously request it. When a user dwells on a widget for a certain amount of time, help is displayed in a popup near the widget. Since hovering over a widget for a few seconds can be an expression of doubt, this technique has potential to assist beginners without bugging experienced users.

Originally tooltips were purely text-based. More recently, certain applications including Word 2010 [64] have started incorporating hotkeys, images and contextual F1 cues (*Figure 2.8*). With the popularization of touch screens, one might wonder if a technique that is so often associated with WIMP interfaces [88] can make the transition to new environments. The main issue is that touch screens are unable to provide a literal translation of the mouse's hover event. Although some academics such as Cheung et al. [13] have tried to find a suitable touch screen equivalent to hover events, these interaction

techniques cannot offer a way to detect hesitation that is as straightforward yet effective. The position and distance of the user's finger relative to the screen are more ambiguous indicators of hesitance. In addition, it may prove challenging to predict which target the user is planning to touch and when. While the hover event could be replaced by a help gesture, such approaches depend on conscious actions by the user and should therefore be classified as "What's this?" help.

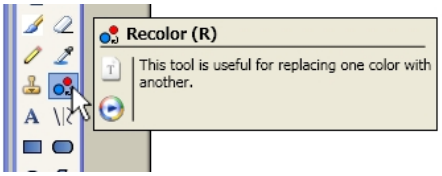


FIGURE 2.9: ToolClips in Paint.NET (source: [29])

ToolClips is a 2010 research prototype by Grossman and Fitzmaurice [29] that augments tooltips in Paint.NET [67] and AutoCAD [3] with new features. In addition to a traditional text description for each painting tool, they also provide access to video demonstrations and full-length help articles. Users can get to this content by clicking buttons inside the tooltip. When one of the buttons is clicked, the popup turns into a floating non-modal dialog that can be moved around. ToolClips do not disappear when the user moves off their parent widget like traditional tooltips do. Instead, they gradually fade out as the user moves away from the popup. Figure 2.9 shows how ToolClips behaves when the user hovers over the Recolor tool in Paint.NET.

The authors conducted a study with 16 paid participants where people were randomly selected to use ToolClips or traditional help. The test focused on high-level Paint.NET tasks that required users to interact with multiple tools. Data analysis revealed that ToolClips users got stuck significantly less often and were able to overcome problems significantly more often. However, there was no sign of a significant difference in subjective satisfaction. A second study was organized to determine whether ToolClips could reduce learnability and retention problems in AutoCAD. During the first session 10 paid participants were instructed to perform low-level tool-based tasks with regular help and ToolClips. The authors reported significantly higher rates of subjective helpfulness for the latter and participants indicated a strong preference towards it. A second session had the goal of testing retention with the same group of participants from the first session. The researchers found that the volunteers completed tasks significantly faster while using ToolClips.

2.2.5 Embodied agents

In 1996, Microsoft first introduced the Office Assistant to Word 97 and other Office apps [55]. It would become one of the most high-profile examples of embodied agents.

This form of support tries to give technology a friendly face by introducing virtual characters such as a talking dog, an anthropomorphic paperclip or a human. The Office Assistant came with multiple characters, but it is perhaps best known for its default persona, Clippy (*Figure 2.10*). The technology behind the characters originated from Microsoft Bob, a non-traditional operating system UI that made intensive use of metaphors and embodied agents [21].

Microsoft created Clippy to fulfill several goals. It served as a gateway to Office's standalone documentation [70]. Users were encouraged to ask questions to the Assistant in regular English, which would theoretically mimic human conversation. In reality, it often failed to interpret the user's intention and people tended to enter one-word questions. Its second goal was to deliver tips to the user that were relevant to their current task. If it detected that a user was taking an inefficient route to complete a task, the Assistant would suggest a faster method. This functionality had its roots in Microsoft Lumière, a project that analyzed people's usage patterns in Excel to provide context-driven assistance [32]. Regarding the Office Assistant, users would often make remarks to Microsoft researchers along these lines: "I am sure there is a better way, but I just don't have time to figure it out" [70]. This finding is in line with the production bias mentioned in Section 2.2.1, which states that users are focused on getting their work done and are not motivated to learn a lot about the system.

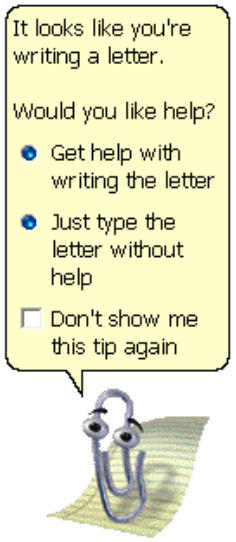


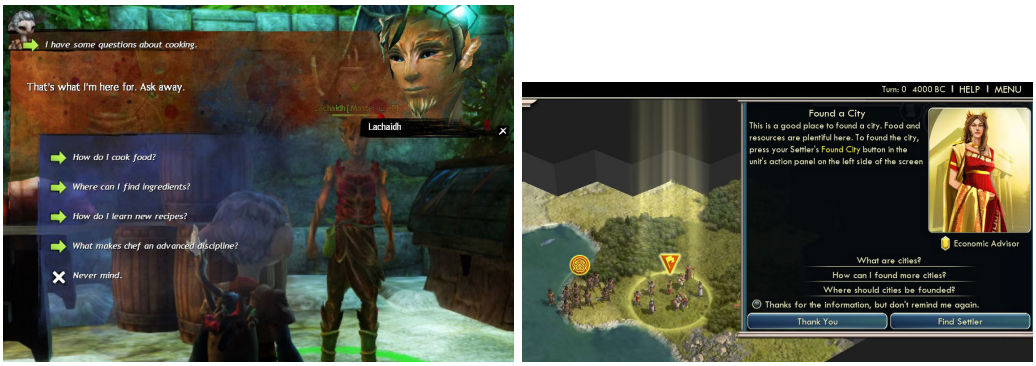
FIGURE 2.10: The last version of Clippy (source: [23])

Clippy was a controversial Office feature. It was perceived as too obtrusive by many users, which prompted Microsoft to tweak the Assistant over the years. A 2007 research paper reported low levels of perceived usefulness and enjoyment of the Assistant among testers [74]. We should note that participants were recruited among students at a North American university where Microsoft Office proficiency was part of the curriculum, so the study may have undersampled inexperienced users. Interestingly, research at Microsoft found that technical people almost uniformly disliked Clippy [70], which leads us to believe that there might be a difference in perception between user groups.

In April 2001, Microsoft launched a tongue-in-cheek ad campaign announcing that Clippy had been fired and that its services were no longer needed for Office XP [51]. Although the Office Assistant could still be brought up from the Help menu, the agent was eventually removed from Office entirely. Microsoft claims that there was roughly a 50-50 split between people who liked it and people who did not, but that this statistic was not good enough for them to keep it around [70]. The technology and the Assistant

characters lived on as a developer API under the name Microsoft Agent for a while, but following the launch of Windows 7 they were discontinued [87].

Although we could not immediately come up with examples of embodied agents in recent software, we spotted them in games with relative ease. Role-playing games such as Guild Wars 2 [30] commonly include characters that provide some sort of interface help or tips (*Figure 2.11a*). The turn-based strategy game Civilization V [15] features a set of adviser characters that help players make informed decisions (*Figure 2.11b*).



(A) Guild Wars 2 (2012) (B) Civilization V (2010)

FIGURE 2.11: Desktop games that incorporate embodied agents

Could it be that embodied agents are a good fit for games, but not for software? Considering that they often integrate smoothly with the game narrative, we suspect that this might be the case. Since software exists to get work done and games are primarily marketed as entertainment, people might have different attitudes towards embodied agents depending on the overall goal of the application. Unfortunately, we could not find any existing research analyzing people's attitudes towards embodied agents in different contexts.

2.2.6 Non-targeted tips

Some applications present tips to the user that have nothing to do with the current context, with the intention of broadening the user's horizons. The earliest form of non-targeted tips that we found was incorporated into Windows 95 (*Figure 2.13a*). Every time the user started Windows, a tip was displayed. This Tip of the Day dialog was adopted by other software from that era, including WinZip [89] (*Figure 2.13b*). With the release of Windows 98, Microsoft removed their Tip of the Day window. As of 2013, WinZip is still using theirs. IntelliJ IDEA [36] implements a subtle variation on the Tip of the

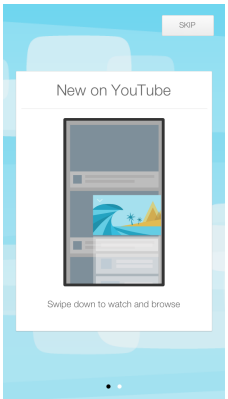
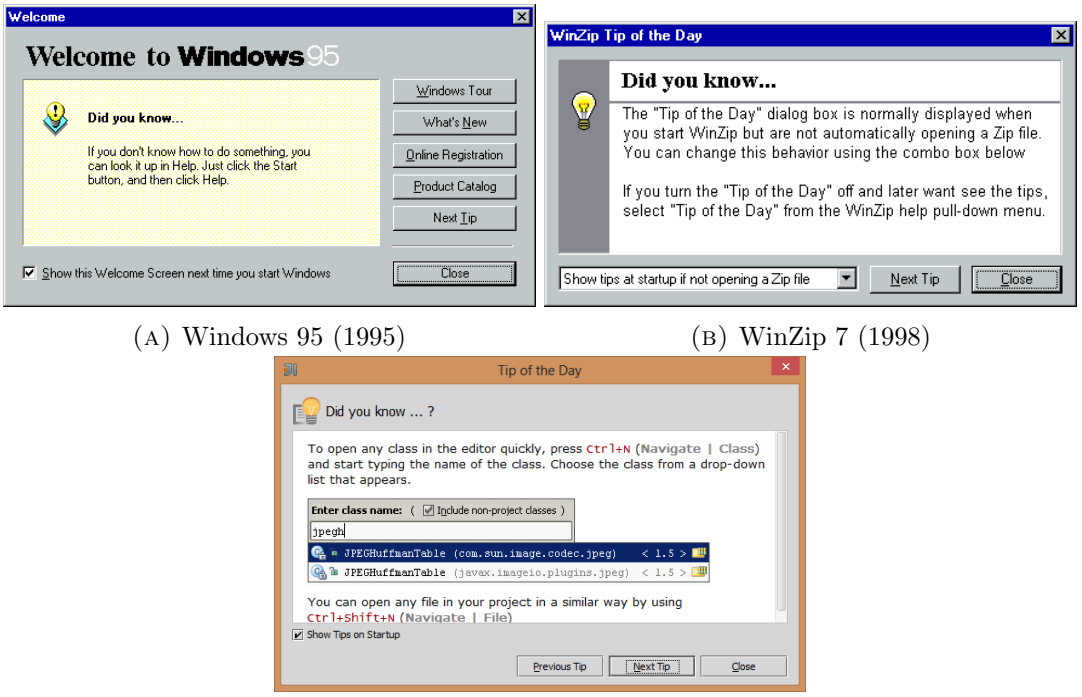


FIGURE 2.12: Announcements of new features (YouTube)

Day window that incorporates screenshots and hotkey coloring (*Figure 2.13c*). More recently, the YouTube app for iOS (*Figure 2.12*) implemented a variation on 'Tip of the Day' windows with big captioned graphics that introduced users to newly added features. Unlike other designs, YouTube's version only shows their tips dialog on first use.



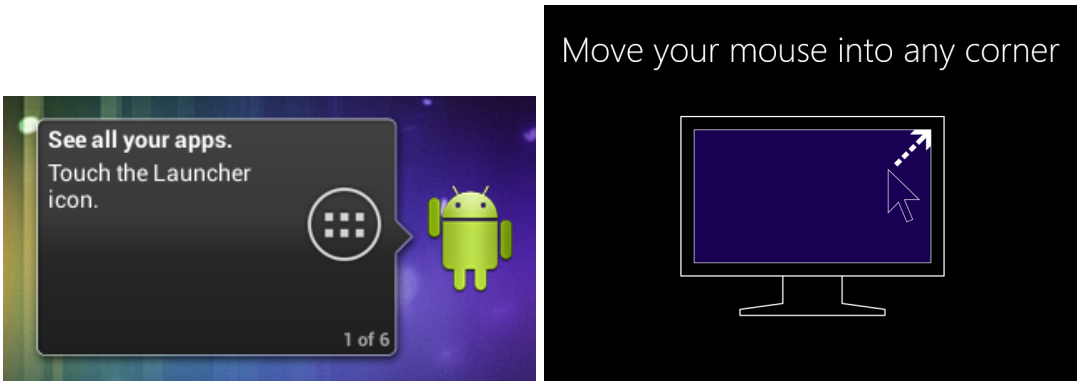
(A) Windows 95 (1995) (B) WinZip 7 (1998)

(C) IntelliJ IDEA 12.x (2012)

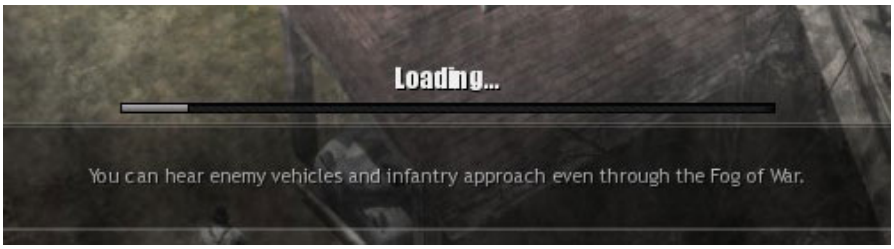
FIGURE 2.13: Various 'Tip of the Day' designs

In theory, Tip of the Day windows might gradually expose users to features that are unfamiliar to them. But are users motivated to read these tips? Will they remember them when they need the information? For reasons similar to those described in 2.2.1, it seems unlikely that these windows help users.

Not all non-targeted tips take on the form of a modal dialog that is shown on startup. Android has been shipping a 'Home screen tips' widget as of version 2.2 [2] to help new users (*Figure 2.14a*). It embeds non-targeted tips into the home screen that do not interrupt the user's task flow. Some games such as Company of Heroes [16] display non-targeted tips on their loading screens (*Figure 2.14c*). At first sight, this looks like a promising method to improve game learnability by making good use of the player's waiting time. The same mechanism has also been applied outside of gaming. In Windows 8, an introductory animation is shown during first boot that explains how its new UI works (*Figure 2.14b*).



(A) 'Home screen tips' in Android 4.2 (B) Windows 8 on first boot (source: [7])



(C) Tips on Company of Heroes' loading screen

FIGURE 2.14: Other forms of non-targeted tips

2.2.7 Usage hints

The intention behind usage hints is to provide instructions explaining how certain functionality works when it is activated consciously or implicitly. It saves users the trouble of consulting standalone documentation if they even bother doing it at all. These hints can reveal additional functions that would otherwise remain unknown and unused.

We found an example of usage hints in image editor Paint.NET [67] (*Figure 2.15*). When users activate the selection tool, a usage hint in the status bar points out that selections can be constrained to a square by holding down Shift. We consider this to be an example of a usage hint rather than embedded help because it is just a quick one-line hint, not a condensed help article.

In gaming, usage hints are often displayed when the user picks up an item. Figure 2.16a provides an example from Marble Blast Gold [20]. When players obtain the gyrocopter powerup for the first time, the following hint is displayed: *"Press the Left Mouse Button to use the Gyrocopter PowerUp!"* When players pick up a propeller hat in New Super Mario Bros. [65], an animated usage hint is shown above Mario's head (*Figure 2.16b*). The animation explains how a Wiimote can be shaken to activate the hat and doubles as a gesture learning technique (*Section 2.2.11*). Usage hints can also be delivered audibly, which is especially useful for games that have a young audience.

FIGURE 2.15: A usage hint in Paint.NET (visual emphasis ours)

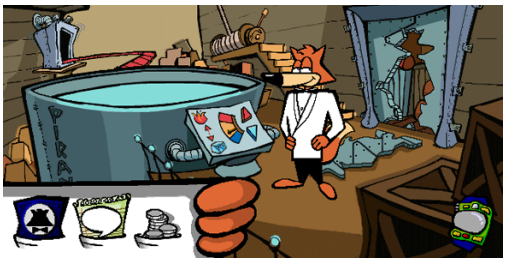
(A) Marble Blast Gold (2003)

(B) New Super Mario Bros. (2009) (source: [59])

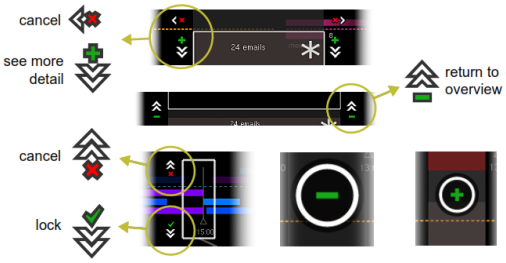
FIGURE 2.16: Usage hints as feedforward in video games

Not only can usage hints serve as a form of feedforward [19], they can also be implemented as feedback. The point-and-click adventure *Spy Fox in "Dry Cereal"* [76] demonstrated this. When players press the blue button on the piranha pool depicted in Figure 2.17a, Spy Fox remarks: *"The piranha pool seems to be getting colder!"* This spoken observation serves as a subtle usage hint to the player.

Although most of the usage hints that we found rely on written or spoken language, it is not a necessity. Flipboard's mobile app [24] uses a peeking animation to demonstrate how users can turn a page. Vogel et al. [84] integrated graphical usage cues for a gestural interface into an ambient display (*Figure 2.17b*).



(A) Spoken feedback



(B) Graphical usage hints (source: [84])

FIGURE 2.17: Various manifestations of usage hints

2.2.8 Feature introductions

Often spotted in games and mobile apps, feature introductions are hints that explain features or game mechanics to new users when they are first encountered. By drawing attention to a feature, an application can reveal functionality that might otherwise remain unknown and unused. An example of this technique can be found in the Flipboard [24] app on iOS (*Figure 2.18*), where it takes on the form of a text popup and a visual highlight. In gaming, AstroPop (*Figure 2.19a*) and more recently Candy Crush Saga (*Figure 2.19b*) use this technique.

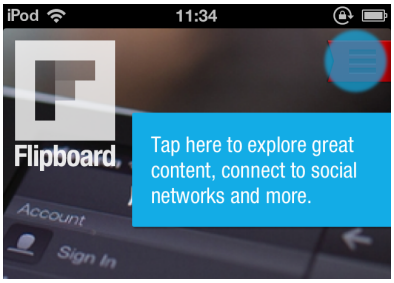
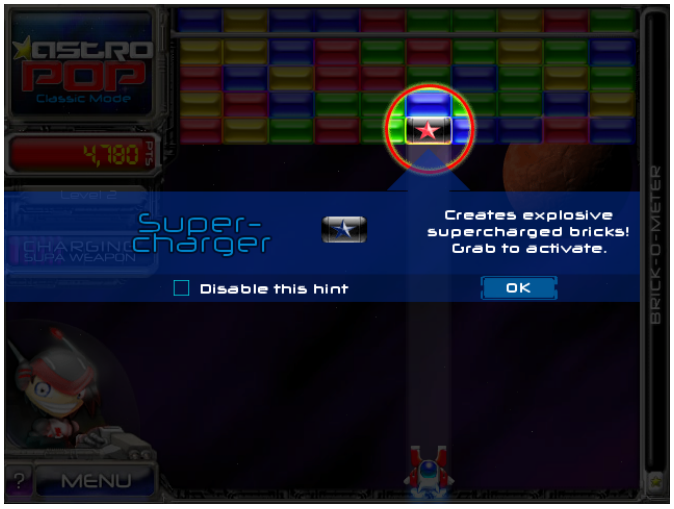


FIGURE 2.18: A feature intro in Flipboard



(A) AstroPop (2004)

(B) Candy Crush Saga (2012)

FIGURE 2.19: Feature introductions in video games

Feature introductions are similar to usage hints (*Section 2.2.7*), but there is a difference in their underlying intentions. While feature introductions are primarily designed to encourage exploration, usage hints exist mostly to help users accomplish existing goals by explaining the currently selected feature. In a sense, feature introductions are also related to the concept of non-targeted tips (*Section 2.2.6*) since they have both been designed to encourage exploration. While feature introductions incorporate some sort of contextual relevance, non-targeted tips are just showing users information that may or may not be relevant to them at some point.

We hypothesize that contextually relevant tips will have a higher recall rate and annoy users less, which leads us to suspect that feature introductions have more potential than non-targeted tips. On the other hand, non-targeted tips may be better suited to

introduce users to features that would normally have low visibility within the application. Then again, if an important feature has low visibility, something might be wrong with the app's UI design. Although this can be an indicator of usability problems, gestural and speech interfaces are exceptions to the rule. Both types of input have inherent visibility issues. Gesture learning techniques (*Section 2.2.11*) and speech learning techniques (*Section 2.2.12*) explicitly try to tackle this specific problem.

Although feature introductions might be helpful to beginners, they can be disruptive to the workflow of an expert. These negative effects can partially be mitigated by letting users skip or disable feature introductions. Figure 2.19 shows two games who have incorporated such a feature. Despite the workaround, experts still have to make a conscious decision to opt out of these intros which is far from perfect.

2.2.9 Action previews

A feature might be easier to understand if the user can see its effect. That is the intended benefit of action previews. A high-profile example of this technique can be found in Word 2010, where users can preview highlighter colors by hovering over a color (*Figure 2.20a*). Similarly, the 'Font' dialog has been using a text field to preview font changes for quite some time. In Windows Vista and up, users can hover on a window in the taskbar to glance at its contents (*Figure 2.20b*). When the user hovers over the thumbnail, they can invoke a full-screen preview that disappears when they remove their mouse from the thumbnail. Note that Windows' thumbnail preview also fits the tooltip category, which makes it a hybrid. Compiz provides a similar thumbnail preview mechanism to Linux users [68]. In terms of Nielsen's heuristics [60], action previews contribute to the 'User control and freedom', 'Error prevention' and 'Recognition rather than recall' categories.

(A) Word 2010 (B) Windows 8

FIGURE 2.20: Action previews in WIMP environments

A problem that can occur with action previews is that the user might be unaware of the change. In terms of our previous example from Word 2010, it is possible that a user

did not notice the change in text background. This phenomenon is known as change blindness. To increase the visibility of such changes, developers might want to draw attention to the affected area by using highlights or animations.

For more complex operations, a summary of changes can be shown. Visual Studio 2012 does this when a developer tries to rename a function or variable, as is shown in Figure 2.21. A confirmation dialog that explains the consequences of a destructive task can also be seen as an action preview.

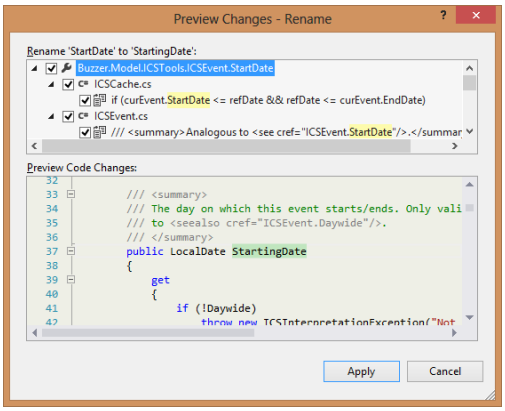


FIGURE 2.21: Preview of complex changes in Visual Studio 2012

Video games frequently incorporate action previews. In simulation games where players can place buildings or objects, an action preview mechanism is often deployed to preview the impact of their actions. Examples include Age of Empires, Rollercoaster Tycoon and The Sims. In Figure 2.22, a player is shown ordering a villager to build a house. Under the build cursor, a preview of the house is shown that looks slightly darker than a real house. When the cursor is hovering over an unusable building spot, a red overlay will be shown over the house. When players find a good spot to build the house, they can confirm their command with a mouse click.

Action previews have also been the subject of academic research. Liang et al. have developed SeeSS [49], a CSS editor that lets designers preview the effect of style changes before applying them. The software is different from the web developer tools in most popular browsers in two ways. SeeSS can evaluate the impact of a change across an entire site, not just the current page. To make it easier for developers to see what changed, SeeSS' change viewer constantly switches between 'Before' and 'After' screenshots using a fade transition, which makes it easier for developers to grasp the difference. On Figure 2.23, the SeeSS editor is shown. Although it only works for static websites (i.e. plain HTML pages), the underlying concept sounds like an effective way to ease the burden of stylesheet maintenance.

FIGURE 2.22: A player is building a house in Age of Empires II: The Conquerors Expansion (2000)

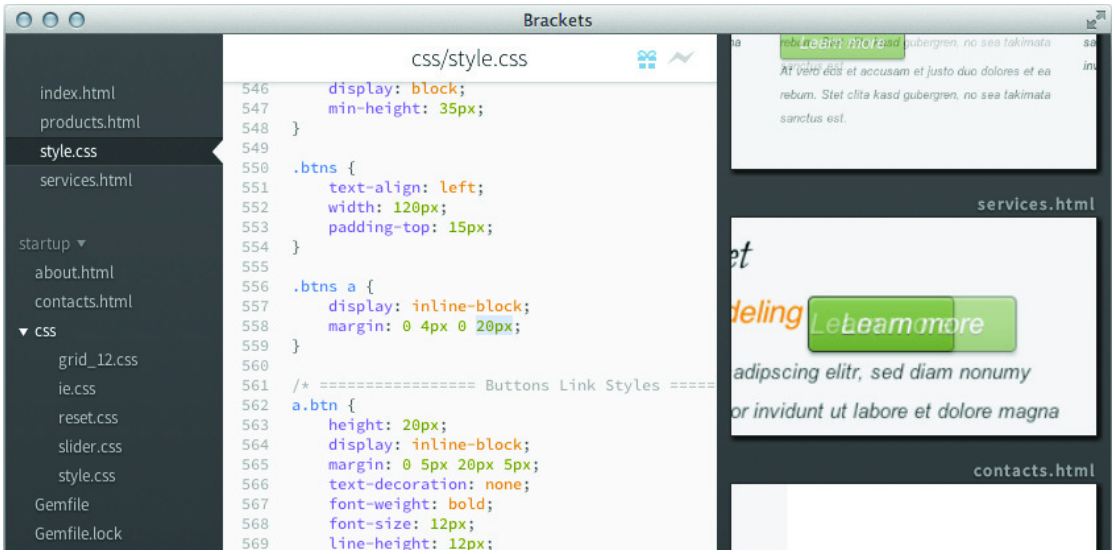


FIGURE 2.23: A CSS editor that uses action previews (source: [49])

Unlike feature intros (*Section 2.2.8*), action previews rarely disrupt the workflow of an expert. They are there when they need it, but they do not have to force themselves upon the user. Then again, feature intros are designed to encourage exploration of unused features; action previews explain features while they are being used.

2.2.10 "What's next?" help

Turn-based strategy games often have complex rules about what players can do during their turn and in which order they need to be done. To help users navigate this structured process, "What's next?" help directs the player's attention to tasks they need to fulfill before they can end their turn. Such help primarily contributes to Nielsen's first heuristic [60], 'Visibility of system status'.

We found this technique in the 2010 game Civilization V [15]. Its interface prominently features a 'Next turn' button that players can use to end their turn. Before they can do so, the game needs certain input from the player. One requirement is that all units must have received orders, but they are allowed to stay on their current tile. Instead of just graying out the 'Next turn' button until the player has met all the requirements, Civilization V replaces it with a button that reads 'A unit needs orders' (*Figure 2.24*). Players can click this button to move the camera to said unit so

FIGURE 2.24: "What's next?" help in Civilization V

that they can provide instructions to them. Additional cues are used to enforce other requirements, such as 'Choose production' and 'Promote a unit'.

We have not yet identified any other instances of this technique. While it can be deployed outside of gaming, it can only work within processes that have fixed requirements. A design tool for a certain type of diagram with multiple constraints could be able to use this technique, for example.

2.2.11 Gesture learning systems

Gesture-based interfaces have a tough time communicating their interaction capabilities to the user. How would a user know that they can swipe their left hand to the right to flip a page using mid-air gestures? When a WIMP interface renders a 'Next page' button on the screen, users know from their experience with buttons that they can single-click the button to flip the page. But if they need to execute a special touch gesture to attach a note to a page, they cannot reasonably be expected to extrapolate from earlier experience or learn all gestures from a paper manual. That is why gesture learning techniques can be helpful in familiarizing users with gestural interface commands. A lot of research has been done on these systems.

While gesture learning techniques often double as usage hints (*Section 2.2.7*), there is a difference. Usage hints announce that functionality can be activated using a certain command (*"Hold Shift to constrain movement to X axis"*), but in most cases they assume that users can already operate the input devices that are involved. A usage hint will rarely tell them where to find the Shift key or how they can execute the press-and-hold maneuver in conjunction with their mouse. Gesture learning techniques explicitly train the user how to execute gestures, as the examples below will show.

Created by Kurtenbach et al. [47] in 1993, **marking menus** are a mechanism that supports the process of learning mouse gestures. To a novice user, they just look and operate like a regular pie menu. But once users start remembering the location of menu items, they can draw the path to the menu item immediately without waiting for the menu to appear. At this point, they have seamlessly made the transition from novice to expert user. If they forget the gesture at any time, they can consult the menu again.

In a later publication [46], researchers implemented marking menus in an application that would be used to perform real work, as opposed to an artificial testing environment. A test was performed with 2 people who used the application for hundreds of hours. One of the conclusions that emerged from the study was that marking gestures were on average 3.5 times faster than pie menu interactions. In Figure 2.26, performance

(A) By using the pie menu (B) By drawing a mark

FIGURE 2.25: How marking menus are operated (source: [46])

speeds of pie menus and marks are compared. The comparison was based on the typical duration of the events that make up both interactions. Although marking menus were designed for pen or mouse input, this mechanism can also be applied to other gestural interfaces.

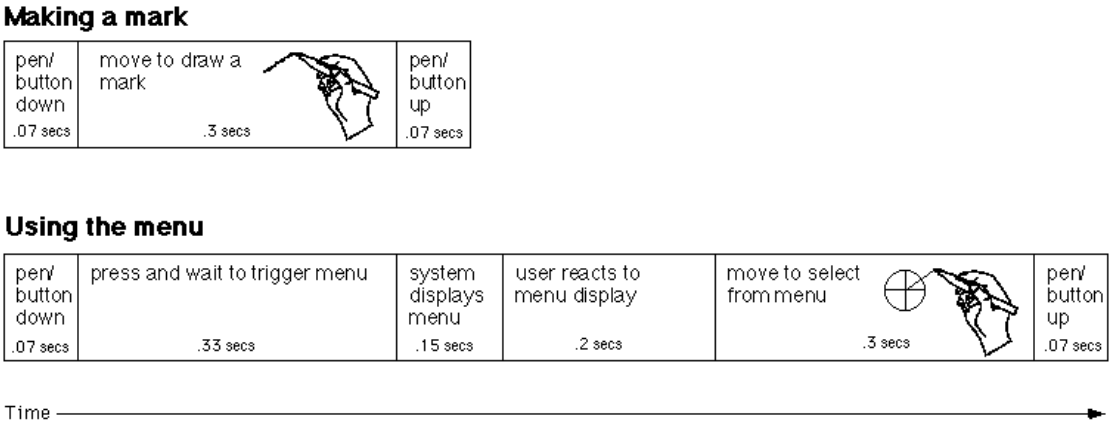


FIGURE 2.26: Typical speeds of marking gestures and pie menu interactions (source: [46])

In 2004, Vogel and Balakrishnan documented how they made an **ambient display** with video-based gesture learning support [84]. When their system picks up on a user's intention to interact, it starts playing a instructional video on loop after a certain amount of time (*Figure 2.27*). In the video, a person executes various gestures. During the demonstration of each gesture, the result of the action is previewed (*Section 2.2.9*). The image of the demonstrator is rendered behind the main UI elements in purple monochrome colors, a design decision that likely reduces its intrusiveness. When the user starts a gesture of their own, the video stops and they regain control over the UI.

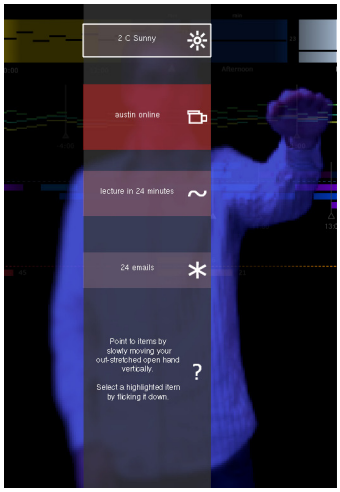


FIGURE 2.27: A looping video tutorial explaining gestures (source: [84])

The system provides one additional form of gesture learning support. A textual hint is shown at the bottom of the screen: *"Point to items by slowly*

moving your out-stretched open hand vertically. Select a highlighted item by flicking it down." Notice how the hint provides detailed instructions explaining how the user can perform the gesture. If the instructions assumed that the user already knew how the gesture worked (e.g. *"Swipe to the left to turn the page"*), they would be a usage hint rather than a form of gesturing support.

In 2008, Bau and Mackay published their work on **OctoPocus**, a system that supports the learning of single-stroke gestures [5]. The idea can be applied to a number of input devices such as touch screens, mice and styli. When the user initiates a gesture, the OctoPocus interface appears after approximately 250 milliseconds. It displays the paths of all gestures the user can perform from their current position. Each gesture path carries a text label. This mechanism has been referred to as *dynamic guides* by the authors. As shown in Figure 2.28, other gestures will slowly disappear as the gesture nears completion. The Achilles' heel of OctoPocus is its visual complexity. While it looks useful for command sets containing a handful of gestures, a larger set would likely overwhelm the user. The authors try to address this problem by (1) making the ends of paths semi-transparent and (2) quickly hiding paths that are no longer relevant.

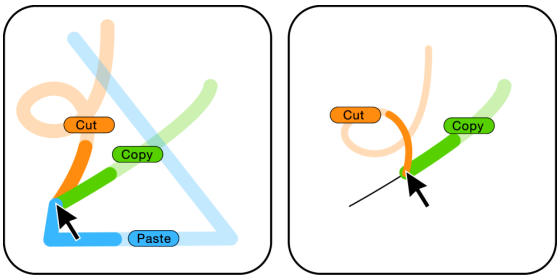


FIGURE 2.28: A gesture learning system for single-stroke gestures (source: [5])

In 2008, Brandl et al. [10] implemented a gesture learning technique for a **bimanual interface** that combines **pen and touch input**. Since users of this system often have multiple ways to invoke an action (e.g. pen/touch, touch/touch or pen/pen), they might have a harder time learning the commands. To support this process, a feature called self-revealing gestures was implemented. When the user initiates an interaction, an overlay in the top left corner displays how the user can complete this interaction (1st row) and which actions can be invoked using these interactions (2nd row). In the example shown in

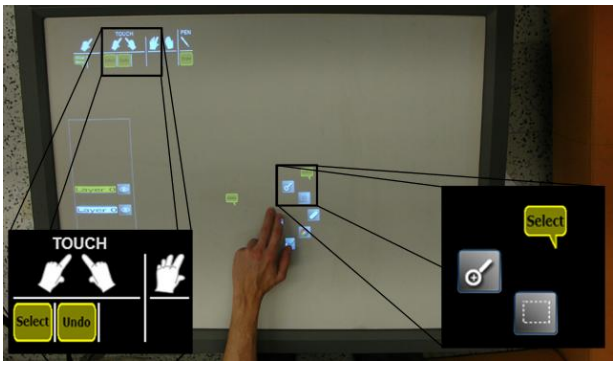


FIGURE 2.29: Gesture completion help in a bimanual pen/touch interface [10]

Figure 2.29, the user has put their first hand down on the screen. If they use their second hand, they can invoke the Select or Undo action. To invoke one of these actions, they need to touch the appropriate text bubble on their screen. In the bottom right corner of the image, the 'Select' bubble is shown as an example. Although the authors used the term tooltip (*Section 2.2.4*) to describe the mechanism, we would primarily classify it as a gesture learning technique because the system's main goal is to describe the gestural interface.

In 2009, Freeman et al. [25] published **ShadowGuides**, which adapts the concept of OctoPocus for multi-touch and whole-hand gestures. This prototype has two major components.

The *user shadow annotations* component serves two functions, one of which is visualizing how the system is perceiving the touch input. In addition, it displays OctoPocus-style dynamic guides with some enhancements. Arrows are displayed for multi-finger gestures to depict the direction of a gesture. For gestures that change shape as they are performed, such as the opening of a fist, the most important steps can be displayed. Dynamic markers coordinate complex multi-finger interactions by telling users when a new finger is needed to complete a gesture, among other things. The *registration pose guide* displays the starting position of the gestures that can be performed.

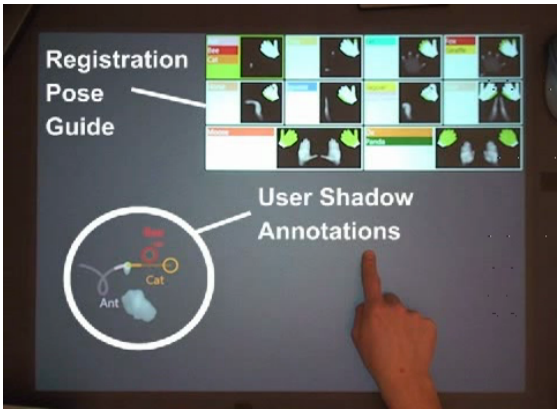


FIGURE 2.30: Screengrab from ShadowGuides' demonstration video (source: [26])

In a test that compared ShadowGuides to video demonstrations, the authors reported that their system performed better on a gesture-memory test and that it got higher ratings for subjective satisfaction. The test was performed with the help of 22 participants who were split evenly between the ShadowGuides and video condition. The authors did acknowledge that video also had its strengths and suggested that future work should investigate whether gesture learning systems could effectively incorporate video.

Released in 2009, **TouchGhosts** by Vanacken [83, 82] explained multi-touch gestures to users by letting virtual hands demonstrate them. Red dots on the hands indicate which

fingers should touch the screen while performing them, as is shown in Figure 2.31. Gestures are demonstrated with the user's own data, not using pre-rendered videos. For example, when the system explains how the user can browse through a pile of pictures, actual photos from the user's image library will be shown. Because these demonstrations also show how the execution of a gesture affects the system's state, they also provide an action preview (*Section 2.2.9*).

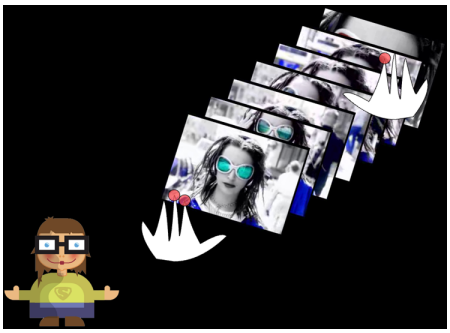
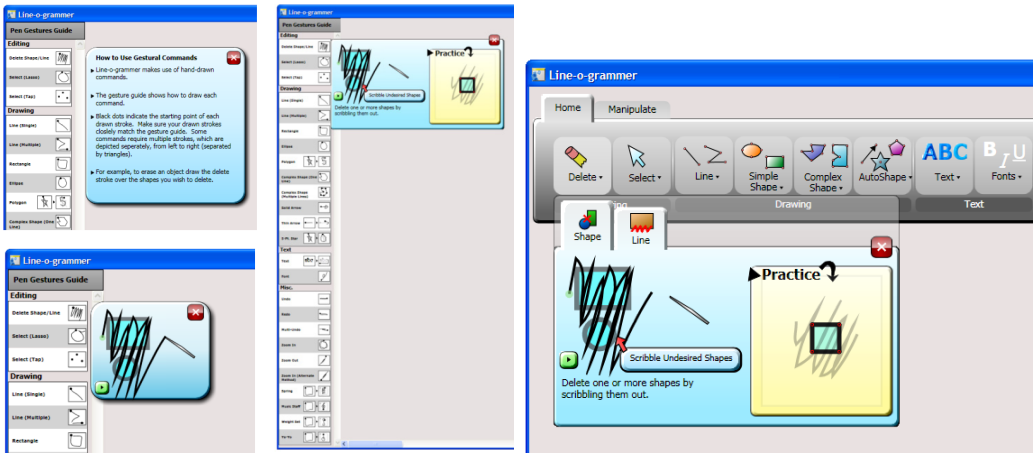


FIGURE 2.31: Virtual hands show how users can flip through pictures (source: [82])

In that same year, Bragdon et al. [9] presented **GestureBar**. Using a Ribbon toolbar like the one in Microsoft Office 2007 [64], it offers a way for users to explore available gestures in a sandbox environment. Its toolbar design facilitates the integration of GestureBar into existing UIs. The core element of GestureBar is the Gesture Explorer, a UI control that can display animations, tips and a practice area for any gesture. The authors evaluated their prototype by comparing it to three other designs in two studies. Their second study had 44 participants create and edit diagrams using one of four systems: CRIB, ANIM, EXPLOR (*Figure 2.32a*) or GestureBar (*Figure 2.32b*).



(A) CRIB, ANIM and EXPLOR (B) GestureBar

FIGURE 2.32: Different UI layouts that the GestureBar researchers tested (source: [9])

- The **CRIB** interface displays a text description of a gesture when clicked. It uses a sidebar layout.
- The **ANIM** interface displays an animated demonstration of a gesture when clicked. It uses a sidebar layout.

- The **EXPLOR** interface displays the Gesture Explorer when a gesture is clicked. It uses a sidebar layout.
- The **GestureBar** interface displays a tabbed version of the Gesture Explorer when a gesture category is clicked. It uses a Ribbon toolbar layout.

Their research showed that GestureBar users were able to successfully execute almost 90% of all gestures, which significantly exceeds the performance of the other three systems. An earlier study with 24 paid participants compared the 4 systems using a within-subject design and indicated that a majority of users preferred GestureBar to find and learn gesture commands.

Arpège by Bau et al. [4] specifically focuses on multi-finger chord gestures. First published in 2010, this technique provides an OctoPocus-style guide for gestures that require users to place two or more fingers on a touch surface simultaneously. In order to implement such a dynamic guide, the researchers had to come up with an abstract graphical representation for gestures, as shown in Figure 2.33. The leftmost image depicts a hand that is executing a 'Save' chord. To decompose this gesture into an arpeggio, each finger was numbered starting with the pinky and ending with the thumb, as shown in the middle. This finger order is used by Arpège to visualize gestures. The rightmost image shows how Arpège's dynamic guide uses this arpeggio. The starting position is highlighted in grey and a thick line indicates what the next finger is.

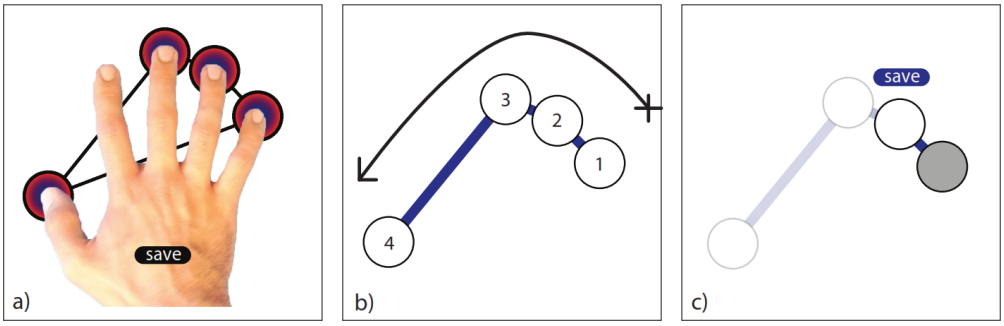


FIGURE 2.33: How a chorded gesture is converted into a representation that Arpège's dynamic guide can use

In Figure 2.34, a user is shown executing a Cut gesture. (a) Initially, 4 gestures are available and their starting points are all shown at once. (b) When the user puts their ring finger down on the Cut circle, the UI registers that they have started an arpeggio. 2 options are currently available: Cut and Copy. (c) The user places their middle finger on the Cut circle and the Copy option disappears. No other gestures are possible at this point. (d) The arpeggio is completed as the user places their index finger on the last circle and the command is executed.

In studies that evaluated Arpège, the researchers found that multi-touch chord gestures were significantly faster than tool palettes, but novice users also made more errors while using the system. In comparison to a cheat sheet, Arpège's performance was not significantly different but it did result in almost 3 times fewer errors. Later work on Arpège explored additional gesture variations such as "tense" and "relaxed" chords [27].

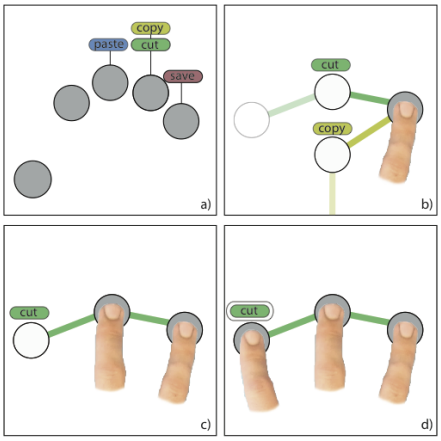


FIGURE 2.34: A 'Cut' chord is executed with Arpège (source: [4])

In **Learning and Performance with Gesture Guides**, Anderson and Bischof [1] discuss limitations to the way that certain gesture learning systems were evaluated by their original authors. They point out that these systems are often evaluated using short-term recall tests that measure user performance, not learning. Although many of the systems discussed in the paper perform well at first sight, users tend to become reliant on them. This reliance impedes the learning process, which causes their performance to take a big hit after a 24-hour break between tests. One could argue that systems like OctoPocus are doing a disservice to their users by making it too easy to blindly follow paths all the time. The authors designed experiments based on common practices in motor learning literature to evaluate four types of gesture learning systems. The tests were performed using single-stroke gestures on a pen-based tablet. Three of them were based on existing systems (*Figure 2.35*):

1. The *crib notes* guide depicted the gestures at half their actual size in the corner of the screen. Participants could not directly compare their trajectory with the reference image and the image did not change as they progressed through a path.

2. The *static tracing* guide displayed an OctoPocus-like path preview under the pen's location at actual size, with the difference that the guide was not updated as the user completed a path.

3. The *dynamic tracing* guide displayed an OctoPocus-like path preview under the pen's location at actual size. Unlike static tracing, this guide updated itself as users progressed through gestures.

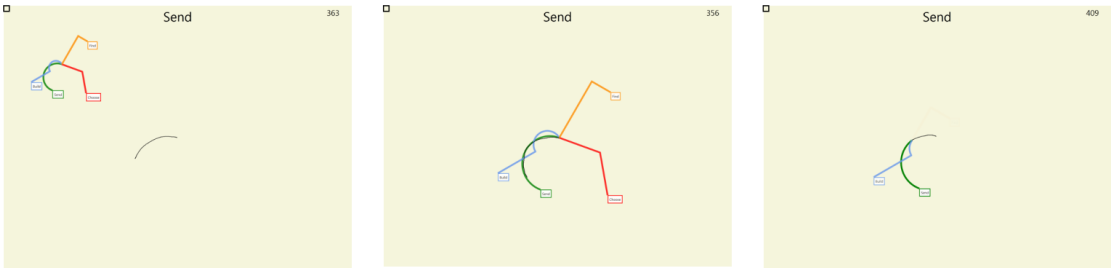


FIGURE 2.35: Three existing types of gesture guides that were tested for learnability. Left to right: crib notes, static tracing and dynamic tracing. (source: [1])

In addition, they derived a new type of guide from static tracing. Their *adaptive guide* gradually reduced the amount of user guidance as the test progressed. Initially, the guide was shown for the entire duration of gestures. Later in the test, the guides starting disappearing halfway through the path. Towards the end of the test, the guides were not shown at all. By gradually phasing out the guide, users cannot keep relying on it and are forced to learn the gestures. Although this negatively impacts performance in the short run, users of traceable guides suffer greater performance losses after a 24-hour break. When users learn the gestures, their performance does not take as much of a hit. In the long term, forcing users to learn the gestures improves their performance.

Forcing users to learn the gestures may be beneficial for additional reasons. If they know their gestures, the guide does not need to be displayed, which reduces visual clutter. Decreased reliance on guides will reduce potential issues related to occlusion. And finally, perhaps users might suffer less of a cognitive burden in the long term if they learn their gestures, allowing them to direct their full attention to the application. On the other hand, infrequent users might be more likely to make mistakes and draw less accurately without guidance. Low accuracy can prove itself to be a big problem for applications with large gesture sets and high gesture similarity rates. However, systems that are optimized for short and infrequent interactions might be better off *with* guidance. For example, one-time users of a public display will not be able to claim the long-time benefits that adaptive guides might bring.

Usage of gesture learning support is not limited to academic prototypes. Windows 8.1 has implemented a mechanism to familiarize novice users with some newly introduced mouse gestures. In Figure 2.36, the user is taught how to open the charm bar. The gesture consists of two steps. After the user moves their mouse into the top right corner, they can move their mouse down across the screen edge to reveal the bar. On mobile devices, users often have to perform a pull-and-release gesture to refresh app content. Twitter's implementation is shown in Figure 2.37.

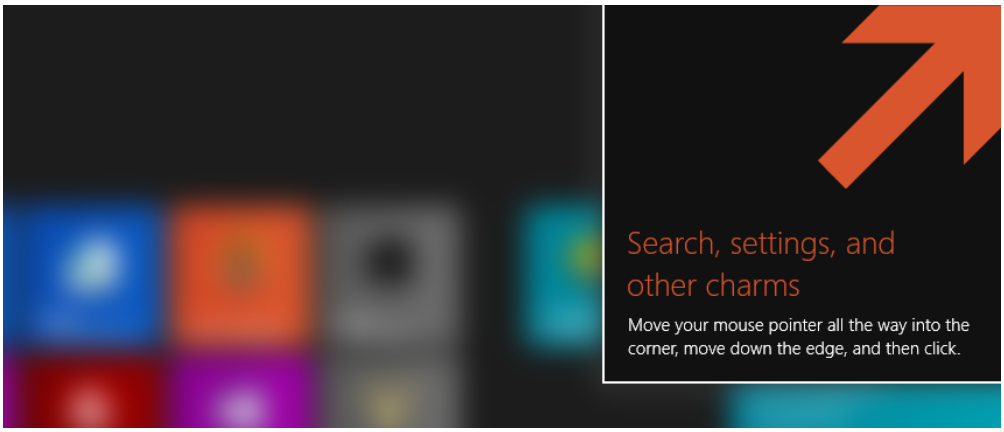


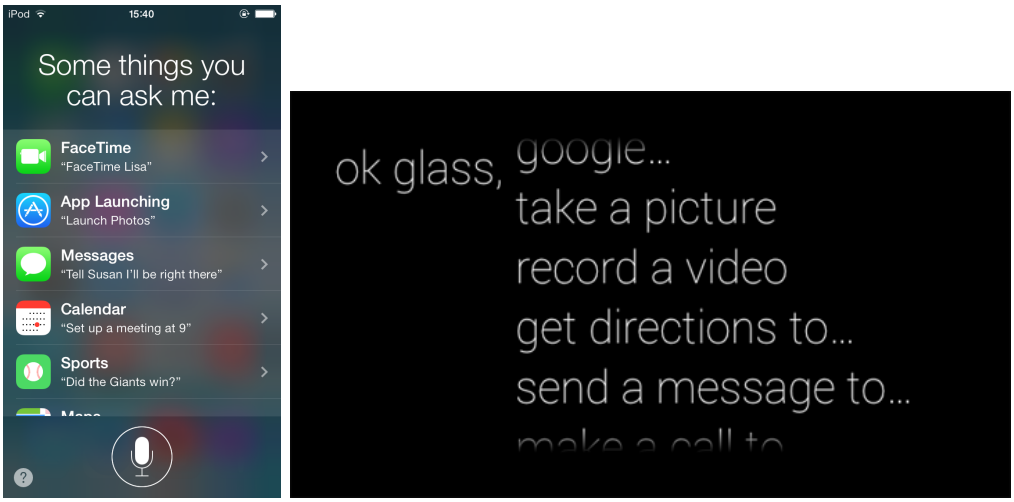
FIGURE 2.36: Gesturing support in Windows 8.1 (blur effect ours)

FIGURE 2.37: The pull-and-release gesture in the Twitter app

2.2.12 Speech learning systems

Speech input and gesture input have similar problems. They have hidden interfaces, so users cannot easily figure out what functionality they offer and how they can access it. Much like gesture learning techniques (*Section 2.2.11*), speech learning techniques aim to make that hidden interface visible. Figure 2.38 provides two examples from commercial mobile devices. Apple's Siri [38] provides a cheat sheet that gives users an idea about the sort of commands that the system can handle. Google Glass provides command suggestions after users activate the speech interface with the 'OK Glass' command.

We will keep our discussion of speech learning systems brief because speech is less established as a form of system input. In addition, speech is less relevant to our later work than other input types.



(A) Siri's cheat sheet in iOS 7

(B) Google Glass giving command suggestions (source: [66])

FIGURE 2.38: Examples of speech learning techniques on mobile devices

2.2.13 Training tasks

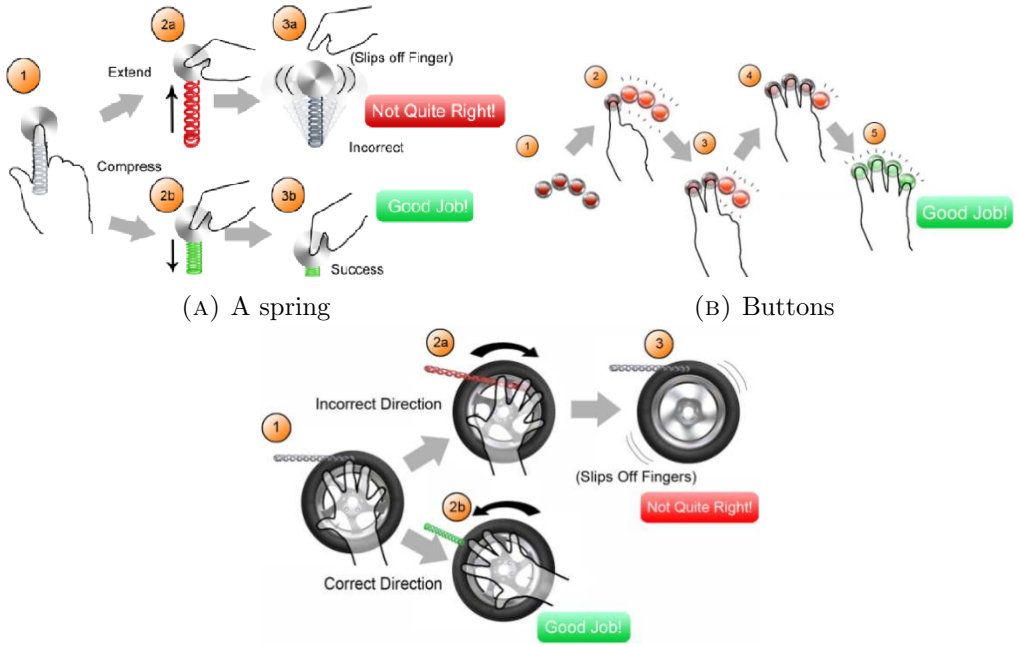
New users can get overwhelmed by the interface of a new application. To reduce the complexity that novices face when they first use it, the application might offer training tasks. By doing this, they can learn the application's interface in a guided sandbox environment. Some applications temporarily reduce the number of visible options in the interface to help them remain focused. Usually, users are asked to complete a predetermined task while hints are provided along the way.

Training tasks are often a natural fit for video games. In Marble Blast Gold [20], the levels from the 'Beginner' set gradually introduce players to various game mechanics like powerups, moving platforms and tornadoes. The first test chambers in Portal [69] teach players how portals work, how the portal gun is operated, how pressure plates function, how they can use momentum to their advantage, and so on. At a later stage, the guidance is phased out as levels become bigger and more complex. These early levels are not explicitly advertised as training tasks. But since they seamlessly integrate with certain games, they can act as a replacement for the game manual without feeling like a pre-game chore.

Training tasks are less common outside of gaming, but researchers have been trying to figure out if there are good ways to integrate them into software. We hypothesize that the reason why training tasks work in video games is that they serve the player's goal: they want to be entertained. And since these training tasks are packaged as entertainment, they fulfill the needs of players while teaching them how the UI works. It is less like that with applications where entertainment is not the primary goal. Referring back to

the production bias described in Section 2.2.1, users are primarily focused on getting their work done and are not motivated to learn a lot about the system. Based on this, we suspect (a) that users are more resistant to training tasks in software and (b) that other techniques might be more appropriate.

Gesture Play by Bragdon et al. [8] is a gesture learning technique (*Section 2.2.11*) that tries to teach gestures through engaging training tasks. The authors' goal was to design games around gestures that would provide enough of a challenge to the user without becoming too difficult, addictive or off-putting. During training, virtual depictions of real-world objects are integrated into puzzles. In the example depicted in Figure 2.39a, the user is asked to compress a spring. By going through the motions to compress the spring, they are effectively executing one of the gestures. Gesture Play uses additional objects like buttons and car wheels depending on the nature of each gesture. Figure 2.39b shows how buttons can be used to design training tasks for chorded gestures without movement. A car wheel can be used to indicate that the user needs to rotate their hand while it is sitting on the screen, as shown in Figure 2.39c.



(A) A spring (B) Buttons

(C) A car wheel

FIGURE 2.39: Examples of the interaction flow in Gesture Play (source: [8])

GamiCAD by Li et al. [48] attempts to ease AutoCAD's steep learning curve by implementing a game-like tutorial system for new users. Their main goal was to provide effective and engaging training tasks. Users of the tutorial system are supposed to help NASA build components for their spacecrafts. The tutorial is split into missions

(*Figure 2.40*) that consist of multiple levels. Each level focuses on a certain task. For instance, one level had this description: *"Use the SELECT ALL feature of TRIM to erase all of the internal lines from a star."* A hints panel that is hidden by default breaks down the process into separate steps and provides feedback on the user's progress. A speed bonus is awarded based on the amount of time the user took to finish a task. Two types of arcade-style bonus levels provide additional training, one of which challenges users to draw the outline of a spaceship as fast as they can.

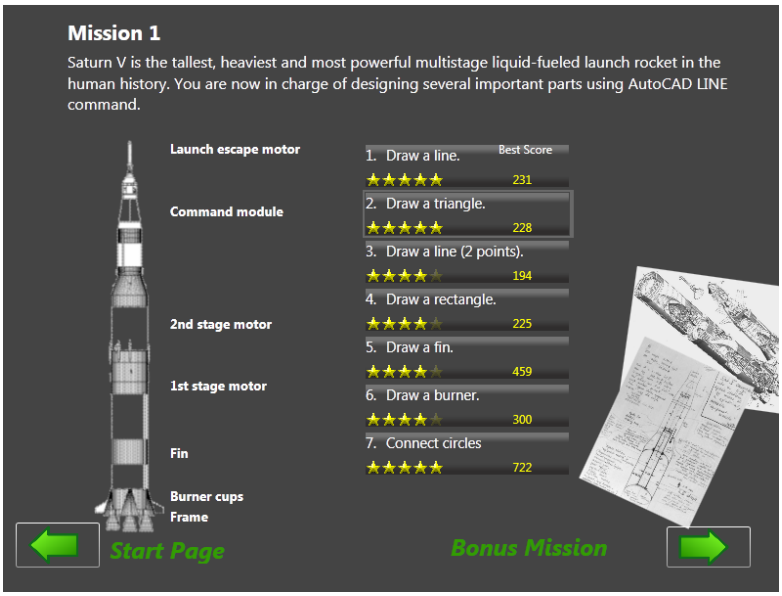


FIGURE 2.40: GamiCAD, a gamified AutoCAD tutorial (source: [48])

Sketch Sketch Revolution by Fernquist et al. [22] is a tutorial system for Autodesk's SketchBook Pro. It is not only designed to familiarize users with the program's interface but also to teach users how to sketch. The system uses drawings by domain experts to guide novices through the process of recreating them step by step. We suspect that since drawing can be a casual activity without a real goal, the production bias will not come into play as much.

2.3 Comparison of techniques

This section will compare the UI learning techniques from 2.2 in a number of ways. Some techniques are inherently limited in the types of input/output that they support. We will also look at their characteristics and the types of help that they can accommodate.

Comparison of UI Learning Techniques

Input Types

	**Keyboard Input** Icon credit: David Cadusseau, from The Noun Project		**Position Control** (e.g. mice) Icon credit: Camila Bertoco, from The Noun Project
	**Speech Input** Icon credit: Travis Yunis		**Rate Control** (e.g. joysticks) Icon credit: Fernando Rojas Braga, from The Noun Project
	**Touch Input** Icon credit: Kenneth Appiah		**Mid-Air Gestures** Icon credit: Megan Strickland

Output Modalities

	**Visual** Icon credit: The Noun Project
	**Auditory** Icon credit: Dirk Unger
Haptic, olfactory and gustatory outputs are outside of the scope of this work.	

Applicability to Various Inputs and Outputs

Technique	Input Types *Green*: supported *Orange*: limited support or less suitable *Red*: not supported	Output Modalities *Green*: supported *Orange*: limited support or less suitable *Red*: not supported
Standalone documentation		
Embedded help		
“What’s this?” help		
Tooltips		
Embodied agents		
Non-targeted tips		
Usage hints		
Feature introductions		
Action previews		
“What’s next?” help		
Gesture learning		

Speech learning		
Training tasks		

Characteristics of UI Learning Techniques

Technique	**Context-aware?**	**Learning payoff** Immediately and/or delayed		**Initiator** User (implicit/explicit) and/or system			**Purpose** Discovery and/or task completion		**Dominant format** Natural language, static graphics and/or dynamic graphics		
		Immed	**Delayed**	**UsrI**	**UsrE**	**Sys**	**Discov**	**TaskC**	**NatL**	**StG**	**DyG**
Standalone documentation	Sometimes	X	X		X		X	X	X		
Embedded help	Yes	X				X	X	X	X		
"What's this?" help	Yes	X			X		X	X	X		
Tooltips	Yes	X		X	X		X	X	X		
Embodied agents	Sometimes	X			X	X	X	X	X		
Non-targeted tips	No		X			X	X		X		
Usage hints	Yes	X		X				X	X		
Feature introductions	Yes	X	X			X	X		X		
Action previews	Yes	X		X	X			X		X	
"What's next?" help	Yes	X				X		X	X		
Gesture learning	Sometimes	Depends		Depends			X	X	X	X	X
Speech learning	Sometimes	X	X	Depends			X	X	X		
Training tasks	No		X			X	X			X	X

Types of Help Supported

Technique	**Procedural information**	**UI instructions**	**Domain information**	**Supplemental information**	**Meta instructions**
Standalone documentation	Yes	Yes	Yes	Yes	Yes
Embedded help	Yes	Yes	Yes	Yes	Yes
"What's this?" help	No	Yes	Yes	Yes	Yes
Tooltips	No	Yes	Yes	Yes	Yes
Embodied agents	Yes	Yes	Yes	Yes	Yes
Non-targeted tips	Yes	Yes	Yes	Yes	Yes
Usage hints	Yes	Yes	No	No	Yes
Feature introductions	Yes	Yes	Yes	Yes	Yes

Action previews	No	No	No	No	Yes
“What’s next?” help	Yes	No	No	No	No
Gesture learning	No	Yes	No	No	Yes
Speech learning	No	Yes	No	No	Yes
Training tasks	Yes	Yes	No	No	Yes

Preview Mechanisms in Gesture Learning Systems

System	**Adaptive?** Contains adaptive and/or non-adaptive elements		**Dominant format** Natural language, static graphics and/ or dynamic graphics			**Domain of gestures covered** Type of gestures supported by each technique
	Adp	**NoAdp**	**NatL**	**StG**	**DyG**	
Marking menus	X		X			Unimanual, 2D, single point
Ambient display		X			X	Unimanual, 2D, single point, with postures
OctoPocus	X			X		Unimanual, 2D, single point
Bimanual pen and touch input		X		X		Bimanual, 2D, multipoint, with postures
ShadowGuides	X	X		X		Bimanual, 2D, multipoint, with postures
TouchGhosts		X			X	Bimanual, 2D, multipoint
GestureBar		X			X	Unimanual, 2D, single point
Arpège	X			X		Unimanual, 2D, multipoint, with postures
Gesture guides	X			X		Unimanual, 2D, single point
Windows 8 gesture tutorial	X		X			Unimanual, 2D, single point
Pull/release to refresh	X		X	X		Unimanual, 2D, single point
Gesture Play	X			X		Unimanual, 2D, multipoint, with postures

2.3.1 Dimensions of the comparison

This section provides additional explanation about the meaning of each table.

- '**Applicability to Various Inputs and Outputs**' explores the potential reach of each UI learning technique across a wide spectrum of I/O devices.

  Green items represent domains of I/O to which a technique can be applied with few adaptations. For example, tooltips are a natural fit for mouse-like devices because they inherently support hover events. *Orange* items represent I/O groups for which a technique might be less suitable. For instance, touch screens do not have an input event that can be a good, unambiguous indicator of hesitation, so touch screens get an orange rating for tooltips. *Red* ratings are used when a technique cannot be applied to an I/O group. Since keyboard and speech input do not support any event that can be an indicator of hesitation, they cannot be used to implement tooltips at all. We will briefly discuss our ratings for each technique in Section 2.3.2.

- '**Characteristics of UI Learning Techniques**' compares groups of learning techniques by analyzing various properties. An explanation for each column is provided below.

  - **Context-aware** UI learning techniques try to provide relevant assistance at the right time and in the right place, demanding as little explicit input from the user as possible. Refer to Section 2.1 for more information.
  - **Learning payoff** looks at the time that it takes for users to benefit from a technique. Many techniques allow users to put new knowledge or skills into practice *immediately*. In other cases, there is a *delay* between when the technique is used and when they can apply what they learned. Information in tooltips is useful right away. A non-targeted tip, on the other hand, may or may not be useful at some point. Some techniques serve up information that can be used right now or at a later point, depending on the current goals of the user.
  - The **initiator** refers to the entity that initiates execution of the UI learning technique. If the *user* makes the conscious decision to get help from this technique, the user is the *explicit* initiator. If the system initiates the technique based on a user action, the user is the *implicit* initiator. If the system autonomously decides to initiate the technique, the *system* is the initiator. Although most techniques have inherent characteristics that restrict who can initiate the execution, it can depend on the implementation.

  - The **purpose** field looks at the underlying intentions behind each group of techniques. The term *discovery* is used for techniques that try to broaden the user's horizons, regardless of their own immediate goals. *Task completion* refers to situations where techniques help users achieve their own pre-existing goals and getting their work done. This often requires some degree of contextual awareness.
  - **Format** looks at whether a technique primarily uses *natural language* (written or spoken), *static graphics* (still graphic elements) or *dynamic graphics* (animations).

- '**Types of Help Supported**' links each category of techniques to the types of help that were described by Rosenbaum et al. [73]. We will provide some more context for these help types in Section 2.3.3.

- '**Preview Mechanisms in Gesture Learning Systems**' compares implementations of gesture learning systems. We devote special attention to this category of UI learning techniques because we will be exploring new systems within this category later in this thesis. More specifically, the table looks at how each system previews gestures.

  - Does the system contain elements that **adapt** to users as they are executing the gesture? For example, OctoPocus is an adaptive system that redraws itself while users are executing a gesture. Although Vogel and Balakrishnan's ambient display picks up on the user's intention to interact, help is more generic and the demonstration does not adapt to the user. In some cases systems can contain a combination of adaptive and non-adaptive elements.
  - What is the **dominant format** that this system uses? Does it communicate with users using *text*, *static graphics* (still graphic elements) or *dynamic graphics* (animations)?
  - Which **domain of gestures** does this system cover? Are bimanual gestures supported? Does it work in three-dimensional space? Does it accept multiple simultaneous input points from one hand? Does it incorporate hand postures?

2.3.2 Discussion of I/O applicability per technique

The '*Applicability to Various Inputs and Outputs*' table examined the flexibility of each technique by looking at the types of I/O that they can be applied to. This section provides further motivation for our coloring decisions in the table.

- **Standalone documentation** – Originally for keyboard and mouse, but can be applied to touchscreen environments. Output is best delivered through a visual modality because it allows for skimming, can carry graphics (both static and animated) and supports hyperlinks.

- **Embedded help** – Similar to standalone documentation, but requires less user input. This makes embedded help more suitable for systems that use unconventional input types.

- **"What's this?" popups** – Works best for input devices that support target selection. Keyboard and speech are less suitable for specifically that reason: they do not provide an intuitive way to let users select the widget they want an explanation for. On the other hand, touch, mid-air, mouse and - to a lesser extent - joystick input give users a relatively straightforward way to select targets. Works visually and audibly because both are suitable to convey short blurbs of plain text.

- **Tooltips** – The most limiting factor with tooltips is the dwelling delay. In mouse terms, the hover event is a great way to detect dwelling behavior. For other input types, it is much harder to find an unambiguous event that is a good indicator of hesitation. Much like "What's this?" popups, tooltips can be delivered visually and audibly.

- **Embodied agents** – There are few inherent limitations to the type of user input that they can work with. It all depends on the nature of the user's interactions. If an agent understands natural language and the conversation flow is open-ended, keyboard and speech input may be preferable. But if an agent uses a fixed conversation flow where users choose from a list of options (*Figure 2.11a*), an input type that supports target selection might be a better fit. Traditionally embodied agents rely heavily on visual output because they have a graphical representation. If we broaden the definition of embodied agents, audio-only implementations can also be considered. For example, oral question-and-answer interactions like those in Apple's Siri [38] are possible under this definition.

- **Non-targeted tips** – Some implementations of non-targeted tips do not require any user input at all (*Figure 2.14c*). For this reason, there are no inherent limitations to the input type. Although non-targeted tips can be delivered audibly, we hypothesize that people might find this too intrusive for a technique that is exploratory in nature.

- **Usage hints** – Minimal implementations do not require any user input. Unlike non-targeted tips, auditory output might be more appropriate because the help is targeted to the user's current situation.

- **Feature introductions** – Minimal implementations only require a way for users to close them. More extreme cases could even close automatically without requiring any user input. As with non-targeted tips, we suspect that audible feature introductions might be more intrusive due to their exploratory nature. However, since feature intros incorporate some level of contextual relevance, users might be more willing to accept them.

- **Action previews** – This technique does not place any inherent restrictions on the type of input that can be used. Although they might work with audio-only output, we think that this output method has inherent limitations that could potentially reduce feedback quality. However, auditory action previews might prove useful in certain niches such as video games for children.

- **"What's next?" help** – No inherent restrictions are in place for the input type. Since this technique depends on visible UI controls, it cannot be implemented solely using auditory output.

- **Gesture learning** – Only works with input devices that support gestures. Keystrokes like Ctrl+A could also be seen as a gesture in the broadest sense of the word. Most input-related limitations are implementation-specific. For example, OctoPocus does not support the chorded gestures that Arpège covers. Since people are heavily dependent on visual and haptic information, we do not see an effective way to implement this technique in an audio-only fashion.

- **Speech learning** – Only applies to speech input for obvious reasons. Audio-only output is a possibility, even though we did not find any examples. In addition, audio-only output cannot provide users a way to skim through a command list.

- **Training tasks** – No inherent limitations are in place for input. Can be implemented using visual or audible output, depending on the type of task. Training tasks for gestures often use detailed visual feedback. A hands-free car navigation system might work better with audio feedback.

2.3.3 Discussion of different help types

The '*Types of Help Supported*' table referred to the five categories of help that Rosenbaum et al. [73] described. The authors identified the following five types of help.

1. **Procedural information** describes how a specific goal can be achieved. For example, Microsoft Word's documentation has an article explaining how users can insert WordArt.

2. **UI instructions** explain how users can interact with a widget. For example, the help function in Google Analytics has an article explaining their date range picker (*Figure 2.41*).

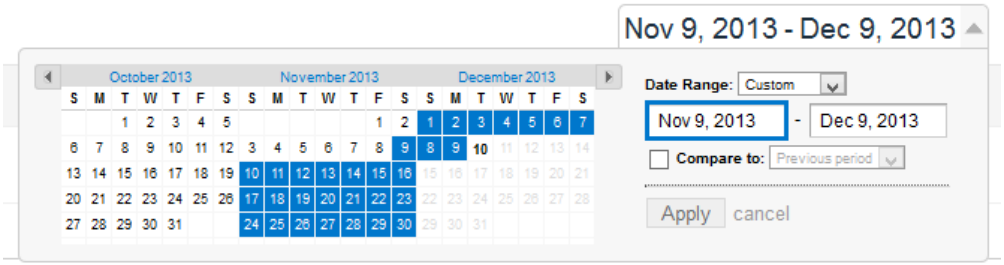


FIGURE 2.41: Google Analytics' date range picker

3. **Domain information** covers help related to the domain of the application. The authors gave the example of a tax filing program that included help on tax codes.

4. **Supplemental information** includes information that can be loosely related to the task at hand, such as a glossary explaining terminology that is used throughout the application.

5. **Meta-instructions** are help that explains how the help system works.

In addition, Rosenbaum et al. make a distinction between functionality support and descriptive support. Functionality support concerns help that is directly related to the user's goals. Descriptive support encompasses help that supports the user's goals, but only in an indirect way. An example regarding a tax filing application was given in the paper. Help articles explaining tax codes usually fall under descriptive support because learning tax codes is not a direct goal for most users. They primarily use the information in the article to get their taxes done. However, if the user is genuinely interested in learning tax codes, it may be considered functionality support. In other words, help can be either functionality support or descriptive support depending on the intentions of individual users. The reason the paper makes this distinction is that functional support should be easily accessible, but it is acceptable for descriptive support require a few extra clicks.

2.4 Adapting to collaborative multiuser environments

Up to this point, we have only discussed UI learning techniques in traditional single-user applications. But what if an application wants to support multiple concurrent users?

Such a setup raises a number of potential issues for some of the techniques that we described.

- **Workspace occlusion** – A technique could interfere with another user's working area, which can diminish that user's sense of control and could negatively affect an application's rating for the 'Error prevention' heuristic [60].

  Solution? An easy way to deal with this problem would be to create private workspace areas for each user. However, this is not always possible or desirable. For example, if two users are creating a drawing on a tabletop, their working areas might move over time as they move around the table to add details.

  Another approach would be to let the system estimate the bounds of each user's working area and use this information to prevent occlusion. But that would require the system to accurately estimate each user's idea of their working area, which is a non-trivial challenge. A feasible heuristic would be to only use the space near the user's last interaction point. This is only possible when the UI uses target selection, e.g. for interfaces that use mouse or touch input. Such a mechanism would be harder to implement for keyboard or speech input. It can be done with mid-air gesturing systems if they work with on-screen pointers, as is the case with the Nintendo Wii.

- **Focus stealing** – Some of the techniques that were described earlier - for instance, some feature introductions (*Section 2.2.8*) - use modal dialogs that block user input outside the dialog. UI learning techniques in systems like TouchGhosts [83] and Vogel's ambient display [84] can temporarily take control of the UI to demonstrate certain features. Although fairly harmless in a single-user environment, it can lead to conflicts in multiuser environments where each individual is focused on achieving their own goals and does not explicitly coordinate everything they do with other users.

  Solution? Developers could work around this problem by avoiding modal dialogs in multiuser interfaces. We realize that this is a radical solution that does not directly address the issue, but there are other alternatives.

  If a system can distinguish the input of each user, it can block input for the user that triggered the modal dialog but not for others. Setups like DiamondTouch [18] (touch input), Dynamo [39] (one mouse for each user) and Carpus [72] (optical hand recognition) have the capability to distinguish input from different users. However, the mechanics behind

selective input blocking might not be clear to the user and that can negatively impact a system's score for the 'Visibility of system status' heuristic [60]. To address this, a subtle feedback mechanism could be designed that informs the user when their input is blocked.

- **Varying levels of experience** – Can a technique accommodate scenarios where a novice and an expert could be using the application at the same time? Since users on each level of experience may require different treatments, techniques should be able to support novices without getting in the way of experts.

  Solution? Some techniques already cater to novices and experts at the same time, allowing for seamless transitions between both user groups. OctoPocus [5] and marking menus [47] do this by waiting a short amount of time before kicking in. This allows experts to execute gestures without going through the guidance system, which saves them the trouble of processing unnecessary information.

  Other techniques, like training tasks, are more difficult to adapt. They often take over the entire application to train novices and cater to the progress of one user, so they are less suitable for multiuser environments. It is not unlikely that there is less of a need for training tasks in collaborative environments because novices can learn by observing expert users.

- **Actions involving multiple users** – Sometimes actions involve input from multiple users at the same time. For example, Morris et al. [57] experimented with gestures that require cooperation from all users. A drawing program they developed had an erase gesture that people could use to correct their own mistakes. But if everyone performed the erase gesture simultaneously, the entire drawing was erased. Some techniques, especially those in the gesture learning category (*Section 2.2.11*), need to be rethought to support multiuser actions.

2.5 Conclusion

In this chapter we looked at 13 categories of UI learning techniques that were invented to complement or replace traditional system manuals. They were extracted from existing software and games as well as academic research. UI learning techniques fulfill various functions. They may assist users in completing their preexisting tasks, help them to discover new functionality or provide additional explanation for more complex UI elements, among other things. We paid special attention to the 'Gesture learning systems' category, which is the category that will be most relevant to our work in Chapter 3.

Chapter 3

Designing a System that Facilitates the Execution of Mid-Air Gestures

Now that we have covered a wide range of UI learning techniques that originated from real-life examples as well as academic literature, we will do further work in the gesture learning systems category (*Section 2.2.11*). Most of the gesture learning systems that we have covered focus on unimanual 2D interaction. Once we venture into less familiar territory, it becomes apparent that existing systems are not fully suitable to support forms of interaction that are less established.

Mid-air gestures is one of these less established domains. How can we clearly communicate a 3D gesture to a user on a flat 2D screen? Do we even use a screen at all? If we think outside the box, we can experiment with alternative ways to communicate gesture paths to users. We will further explore the design challenges of mid-air gestures in Section 3.1.

After an initial exploration, we develop a prototype of a gesture learning system specifically aimed at supporting the process of executing stroke-based mid-air gestures. In Section 3.2, we describe its behavior, its limitations and some practical applications. To bridge the gap between artificial testing environments and real-world usage, Section 3.3 describes how we envision our visualization being used in real applications.

3.1 Challenges of supporting mid-air gestures

Section 2.2.11 discussed various gesture learning systems that already exist. But when implementing mid-air gestures, developers are faced with a number of unique challenges not found in more established input technologies, such as multitouch.

- **More problems with visual clutter**. One of the main limitations in OctoPocus [5] is the high amount of visual clutter caused by an abundance of options at the start of a gesture. The researchers have tried to address this limitation by only showing the nearest part of each trail at full opacity and by quickly eliminating gesture candidates with low confidence scores.

  Adapting the concept of OctoPocus to accommodate 3D gestures would likely prove itself to be more difficult. The addition of a third dimension introduces more occlusion issues. How will users be able to deduce that a gesture is supposed to start by going forward if other gestures are occluding its path? Section 3.2 will explain the clutter reduction measures that we implemented in our prototype.

- **Visualizing 3D**. How can we communicate the shape of a 3D gesture path on a 2D screen? There are several optical tricks that let us convey "fake" depth information using depth cues. Such depth cues include motion parallax, lighting differences, drop shadows, perspective and blur effects based on distance from the camera.

  We could also avoid traditional displays altogether and instead take a different path. LightGuide [75] by Sodhi et al. is an augmented reality system that uses a Microsoft Kinect [40] depth camera and a projector to render visualizations on users' hands. It was designed to help athletes, injured patients and musicians master movements without the presence of an instructor. One of its visualizations is a 3D pathlet (*Figure 3.1*), which indicates to the user where they are supposed to move. A drop shadow is rendered underneath the pathlet as a depth cue. Although LightGuide is not a gesture learning system, it reminded us to think outside of the screen. Other avenues worth investigating include 3D-enabled HMDs with augmented reality features and stereo displays.

  FIGURE 3.1: LightGuide's 3D pathlet visualization (source: [75])

- **Detecting start and end points for gestures**. Mid-air input does not have an obvious event that could indicate the start of a gesture. This stands in contrast to many other input methods, such as mice ('left button down' event) and touch

screens ('finger down' event). But mid-air input sensors are continuously picking up information, which blurs the line between irrelevant movement and intentional system interaction.

One way to tackle this problem is to require users to assume a special full-body posture that can reliably be recognized by the system. Walter et al. played with the idea of a teapot gesture in StrikeAPose [86]. When users want to interact with the screen, they use their arm to create an enclosed area near their hip as shown in Figure 3.2. This solution is not very subtle. Users might feel a little embarrassed when assuming such an unconventional posture, especially in public spaces and outside of entertainment applications. A second limitation of this approach is that it can only be applied to unimanual gesture sets because it only leaves users with one free hand.

FIGURE 3.2: StrikeAPose's teapot gesture (source: [86])

One possible solution would involve users signaling their intention to perform a gesture by raising their hand near their shoulder and then assuming a predetermined hand posture. The end of the gesture can be signaled by releasing the posture. Although this approach sounds reasonable in theory, data from mid-air gesture trackers was not detailed enough to implement such a technique at the time of writing. First-generation Kinect hardware can track the position of several types of joints, but is unable to distinguish fingers [79]. Since this problem is outside the focus of this thesis, we will work within current tracker limitations to implement a set of mid-air gestures.

3.2 Prototyping a mid-air gesture learning system

We implemented a prototype of a gesture learning system that specifically targets mid-air gestures in three-dimensional space. It was written in C# and relies heavily on Windows Presentation Foundation features. Much like OctoPocus, it guides the user through the process of executing stroke-based gestures. Our system supports both discrete and continuous 3D gestures. In this section, we will discuss the details of our implementation and look at various interaction concepts that we experimented with.

The prototype went through several iterations and evolved gradually. Major changes to the prototype will be discussed throughout this section where appropriate. The system underwent additional changes after our pilot study. We will discuss these changes separately in Section 4.2.

3.2.1 Gathering user input

We used a first-generation Kinect camera to implement our prototype. We worked with version 1.8 of the official Kinect SDK [41], which was the current version at the time of implementation. The SDK implements a skeletal tracking feature that provides location information about users' joints. It can track the locations of up to six people in 3D and provide detailed joint positions for up to two people. Figure 3.3 shows the joints that can be tracked by the Kinect camera. In our prototype we used the left hand, right hand, spine and head.

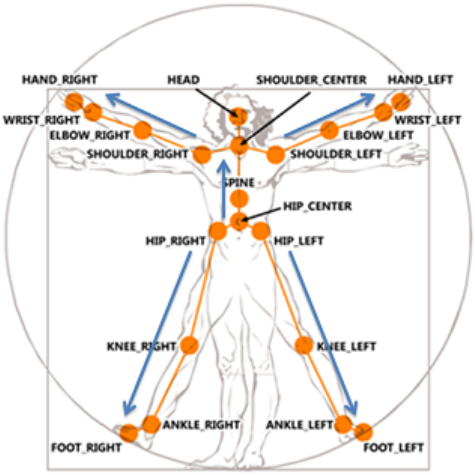


FIGURE 3.3: Overview of the trackable joints (source: [80])

3.2.2 Tracking hand movements

As discussed in Section 3.1, technical limitations to our camera made it hard to come up with a seamless way to activate hand tracking. Initially, we turned on hand tracking a few frames after the user's hand rose near their shoulder. This approach would often introduce an unwanted upward stroke at the beginning of our gesture path, so it was not a reliable way to deduce what the user considered to be part of the gesture. We solved this during early stages of development by using the left hand to activate the system and using the right hand to draw the gesture. The procedure was as follows:

1. Raise right hand and hold still
2. Raise left hand to indicate start of gesture
3. Trace path of gesture that needs to be executed
4. (Optional) Cancel gesture by lowering left hand before path is fully traced

Although this mechanism would work fine for right-handed unimanual gestures, it proved unsuitable because we also wanted to support bimanual gestures and left-handed unimanual gestures. That prompted us to implement a new activation mechanism that did not rely on hands. We used bone orientation data from the Kinect's skeleton stream to detect when users tilted their head to the right. This is how users were supposed to execute gestures after the implementation of the 'head tilt' event:

1. Raise left and/or right hand above spine
2. Tilt head to the right to indicate start of gesture
3. Trace path of gesture that needs to be executed
4. (Optional) Cancel gesture by tilting head to the right

When users raise one or two hands, an overview of respectively unimanual or bimanual actions is displayed. These gesture overviews serve two purposes. First of all, they help users to find the command they are looking for. Secondly, they also indicate how many hands the system is detecting at that particular moment.

Note that users are not required to keep their head tilted while executing the gesture since that would significantly undermine the usability of the system. The interaction was implemented as a press-and-release operation rather than a press-and-hold operation. To help new users find their way with the interface, we implemented a few animated usage hints to guide new users through the activation process (*Figure 3.4*). Later versions of the visualization also added a cancellation hint that was displayed during gesture execution (*Figure 3.5*).

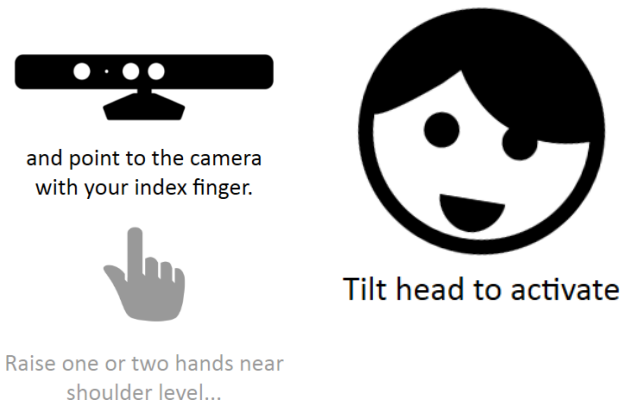


FIGURE 3.4: Animated usage hints for new users

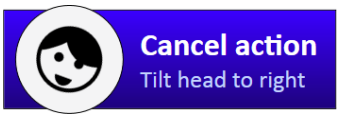


FIGURE 3.5: Gesture cancellation hint

3.2.3 Performing gestures

When the user initiates a gesture and dwells near the starting point, an overview of all gestures is presented to the user. From this overview they can deduce which actions are available and in which direction their gesture paths start. This allows us to reduce the number of concurrently shown gesture paths and therefore reduce clutter issues that were mentioned in Section 3.1. Initially, we displayed the gesture overview that was also shown before activation (*Figure 3.6*).

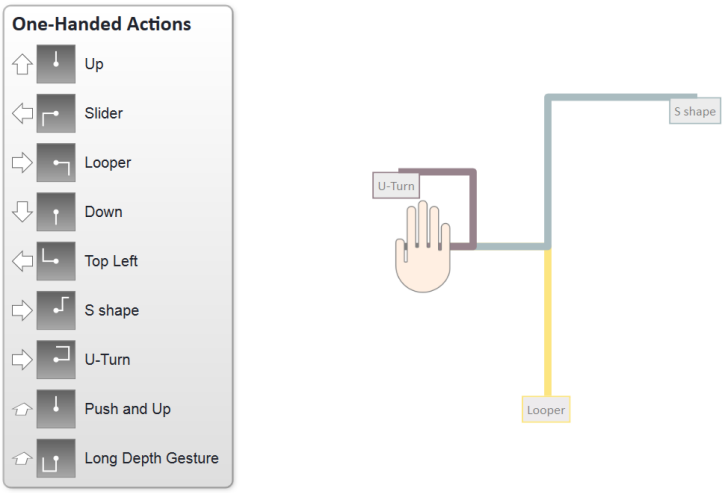


FIGURE 3.6: Old starting visualization

In later versions we replaced this overview with a more integrated display that brings the gesture names and their initial direction into the visualization itself (*Figure 3.7*). As users move closer to any of the directional arrows, their size starts increasing and their fill color starts transitioning to black. After the user hits the arrow, the full paths of gestures going in that direction are revealed and other gestures disappear. We believe that the new starting visualization better communicates the full set of available starting directions, all while keeping clutter at a minimum.

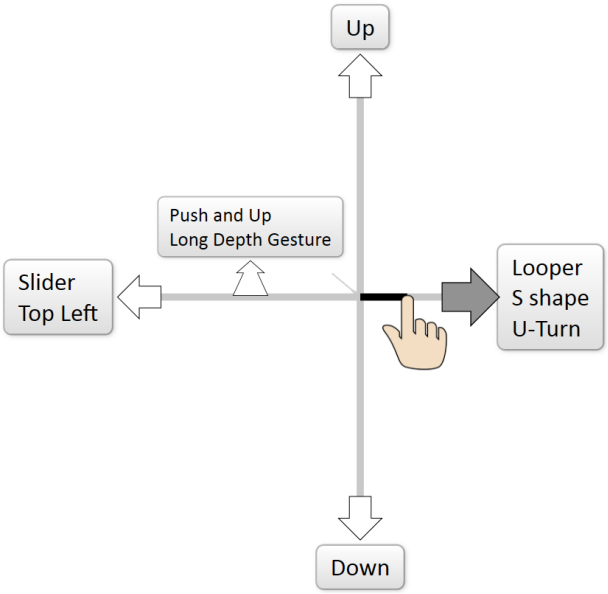
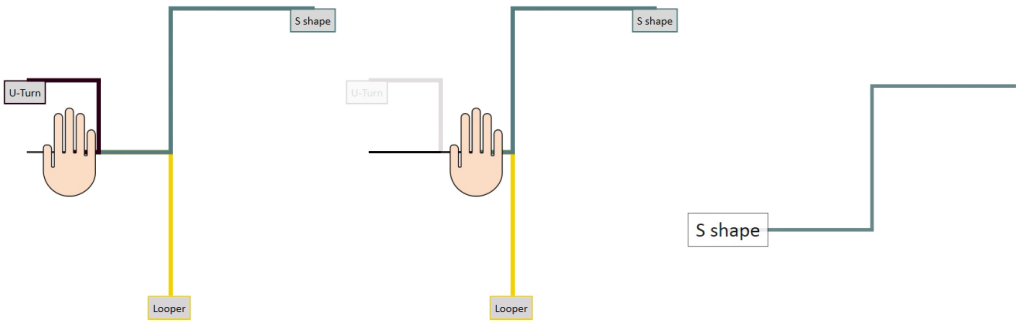


FIGURE 3.7: New starting visualization

The user's hand is represented by a pointer during the whole process. As the pointer moves towards a specific path, ineligible candidates are gradually eliminated and their paths are faded out until they are no longer visible. When the user finishes their stroke, a text label in the middle of the screen will confirm that their gesture was recognized. To accentuate this textual feedback, its font size gradually grows while the label's opacity gradually fades to zero. The whole process is illustrated in Figure 3.8.



(A) User starts stroking a gesture path (B) Candidates fade out as they lose relevance (C) Feedback after user finishes stroke

FIGURE 3.8: The process of drawing gestures

The user's hand pointer always snaps to the closest gesture path. This decision was made to facilitate input, to reduce clutter and to mask inaccuracies in the Kinect's skeletal data. Snapping also made it easier for us to implement a simple gesture recognizer (*Section 3.2.5*). The principle of our basic snapping algorithm is shown in Figure 3.9.

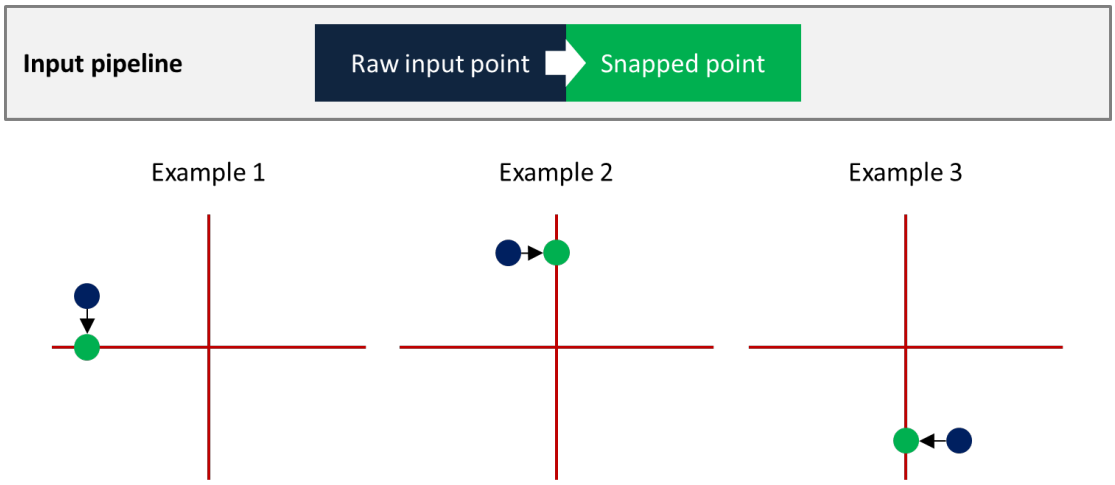


FIGURE 3.9: Basic snapping behavior

The downside of our basic snapping behavior was that it would sometimes lead to unexpected and undesired jumps in the hand pointer's position. To prevent sudden jumps, we introduced offsets that loosely constrain the user's movements to a tunnel. If users move their hand to the left while their pointer is on a vertical line, their pointer will not make a sudden jump as Figure 3.10 explains.

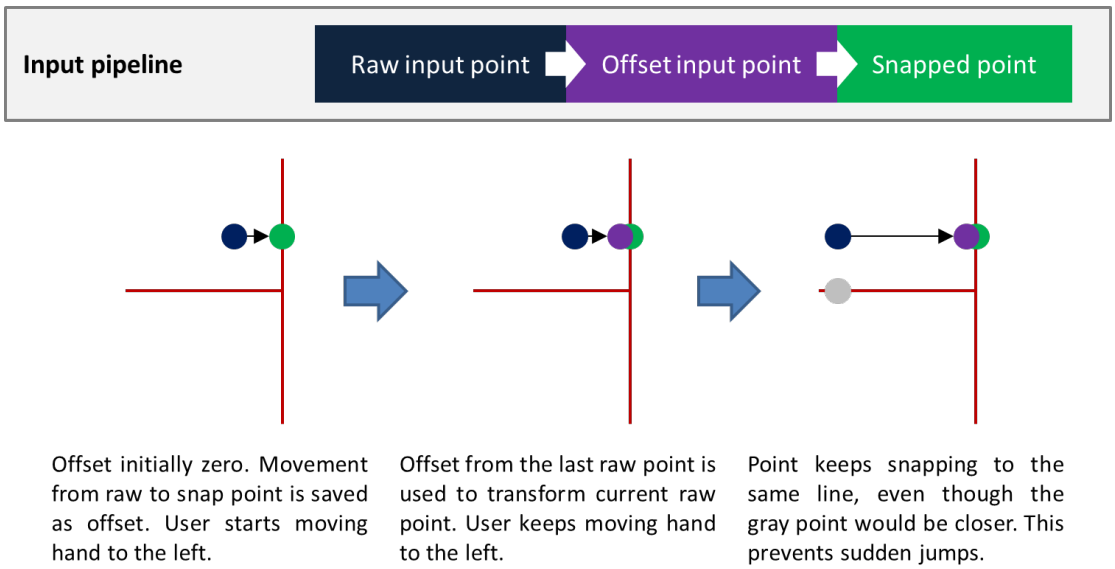


FIGURE 3.10: Offset snapping behavior

After the introduction of offsets, it became harder to make turns at intersections. This unwanted side effect popped up because our new input constraints were designed to prevent sudden jumps. In later versions of the prototype, we tried to address this shortcoming by developing the concept of *diamond snapping*. The idea behind diamond snapping is that constraints on user input are loosened when the hand pointer is near a

junction. Movements near junctions are constrained to a diamond shape using the rules that are demonstrated in Figure 3.11. Hand pointer movements are smoothly guided into one of the intersecting paths along the diamond's border.

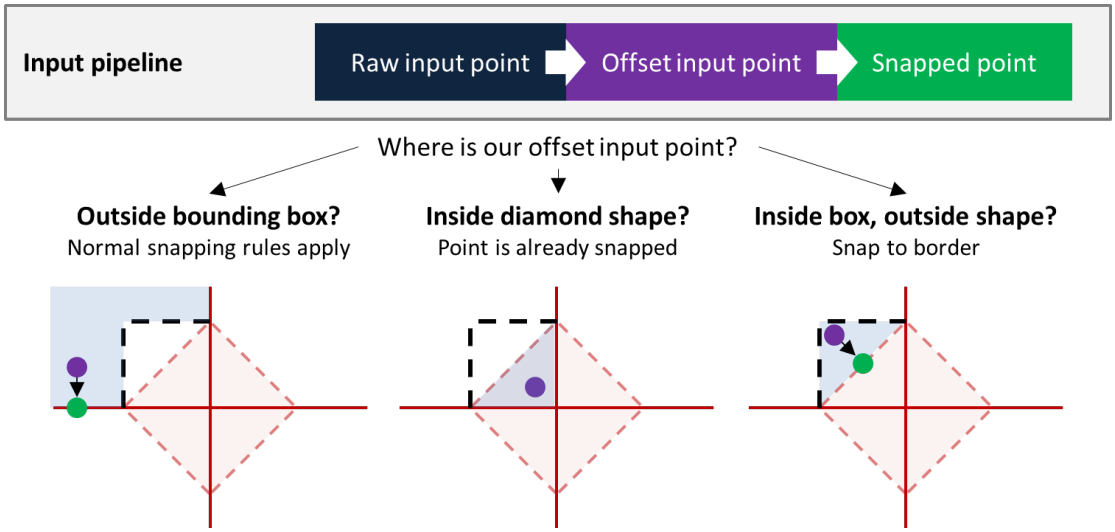


FIGURE 3.11: Simplified explanation of diamond snapping rules

3.2.4 Different gesture performance modes

We experimented with three operating modes for the gesture recognizer in early versions. Each of these modes implement minor variations in the way that users can interact with our system.

The **'Undo' mode** allows users to backtrack on their path if they accidentally went the wrong way, restoring gesture candidates that were previously eliminated. To support the restoration of eliminated candidates, we implemented a stack that stores previous states of the recognizer (*Figure 3.12*). Every time the user reaches a junction[1] or a bend point, a copy of the current state is pushed to the stack right before any candidates are eliminated. When the user returns to the last junction or bend point, the last state is popped from the stack and any candidates that were eliminated here are once again in the running.

[1] Junctions are points where multiple gesture paths diverge for the first time.

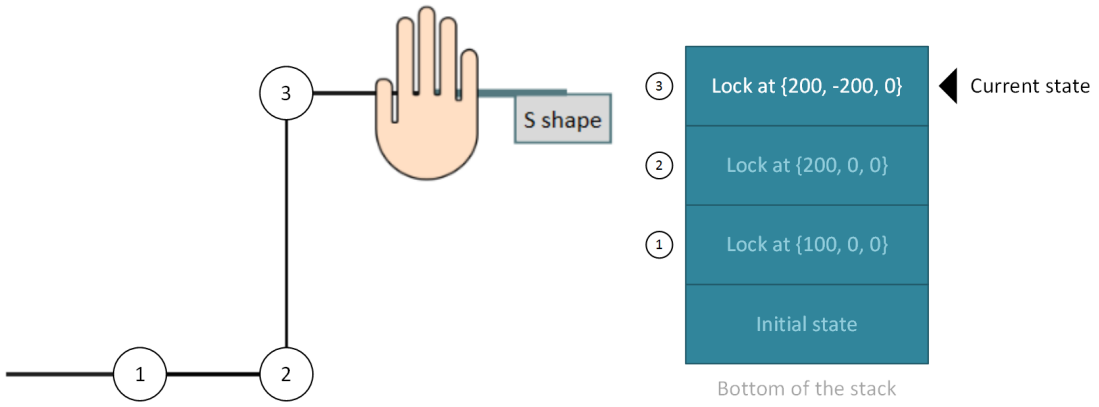


FIGURE 3.12: The unimanual undo stack

The **'Lock' mode** does not allow backtracking past the last bend point. This used to be the default mode before the undo stack was implemented. A padlock icon is shown at the lock point to provide users with feedback (*Figure 3.13*). When the lock point changes, the user's attention is drawn to it using a scale animation. It might be helpful to use locking on long gesture paths if they require lots of movement or if the sensor's accuracy is lacking. By locking the path in situations where the user is unlikely to backtrack, we might help them get to their goal more quickly. But perhaps more importantly, it impedes the user's ability to undo their actions if they decide to go a different way. In terms of Nielsen's heuristics [60], it harms user control and freedom. If the paths are short, users might choose to cancel and restart by executing two consecutive head tilts, but that is far from ideal. We could improve the locking mechanism by giving users a way to "break" the lock if they need to. A relatively straightforward way to implement this is to have the lock disappear if users apply enough force to it with the hand pointer.

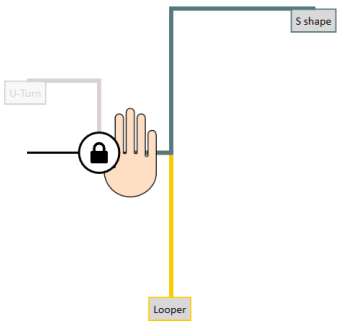


FIGURE 3.13: 'Lock' mode

The **'Cancel' mode** provides users with an additional way to cancel a gesture before completing it. *(As explained in Section 3.2.2, users can already cancel a gesture halfway by tilting their head to the right.)* By moving their hand towards the cancel icon at the second-to-last bend point (*Figure 3.14*), they can exit the gesture without executing any action. Providing a complementary way for users to cancel gestures might improve user control and freedom. However, as is the case with locking, this mode interferes with the user's ability to backtrack on their path. A lock

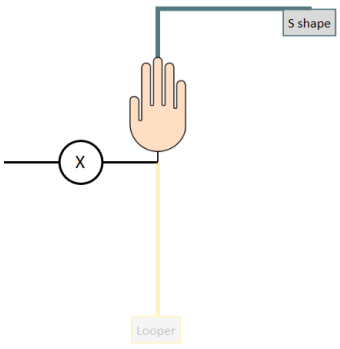


FIGURE 3.14: 'Cancel' mode

breaking mechanism will not be able to help this time, given that the 'hand touches icon' event is already used to cancel the gesture. We suspect that users are more likely to backtrack if the average gesture path is longer, so the 'Cancel' mode might be appreciated more by users when paths are relatively short.

Later versions of the prototype all used the 'Undo' mode. We preferred this mode over the others because neither 'Lock' nor 'Cancel' allowed users to backtrack on their path. Additionally we hypothesized that the amount of accidental cancellations would be higher for the 'Cancel' mode than for the 'Undo' mode. And since the 'Cancel' mode provided redundant functionality that was also offered by the activation mechanism, we concluded that 'Undo' would be most useful to our users.

3.2.5 Recognizing gestures

Initially, we worked with a two-dimensional gesture recognizer by Kristensson and Denby [44]. It was originally made to recognize pen strokes and touch-screen gestures. The authors also provided a Java implementation of their algorithm that could be plugged in to our existing C# codebase using IKVM.NET [34]. When supplied with a list of gesture templates and a user-drawn path, the recognizer returned a list of possible matches along with their confidence rating. We ran into a couple of issues with this recognizer.

- Since mid-air gestures can be three-dimensional, we needed a way to extend the existing recognizer's reach to 3D. We tried projecting each gesture into the XY and YZ planes and combining confidence ratings with appropriate weights for each plane projection, but results were unpredictable and unsatisfactory.
- The recognizer was unable to tell when a user had finished stroking the entire gesture, which forced the prototype to prematurely declare a gesture as the winner. To illustrate the problem, let us assume a simple gesture set consisting of a left, right, upwards and downwards stroke. When a user accidentally brushed the path of a simple upwards stroke (even slightly), the recognizer would be very confident that this was the gesture the user wanted to draw. We had no way of knowing whether the user actually wanted to go upward or whether it was just noise. (The noise could be caused by an inaccurate sensor reading or due to the limitations of the human motor system.) Although there are some ways to work around this limitation, they were not desirable.
- Manual conversions had to be made between C# and Java data structures, which had the potential to introduce time-wasting bugs.

We ended up moving away from this recognizer and built our own. Our recognizer provides the following advantages over the library that we previously used.

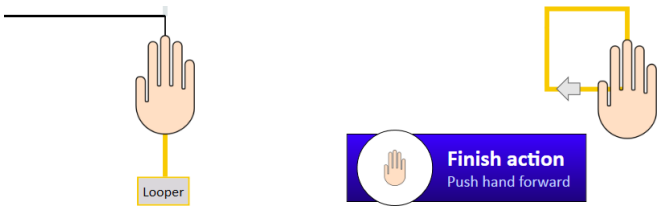
- The recognizer has **native** support for **3D spaces**. There is no need to resort to ineffective workarounds involving plane projections, which increases recognition accuracy.
- The recognizer is more suitable for **live mid-path feedback** as it can detect when gestures have been fully stroked.
- Hand movements are "straightened" by pushing the hand pointer into the nearest eligible gesture path to **reduce the cognitive burden** of executing gestures. We refer to this practice as snapping. We hypothesize that straightened input paths will reduce the time that users need to process the visualizer's feedback.
- Snapping also **increases the accuracy** of the recognizer, meaning that users will have to recover from accidentally executed gestures less often. This will also make the visualization more predictable, which is beneficial to the user.
- The recognizer **eliminates redundant input points** from the path to ease the burden on the recognizer and the visualization.
- Users can **backtrack** during the drawing of a gesture path if they make a mistake or change their mind. This would not be possible if we did not straighten the path. Support for undo raises our score for the Nielsen heuristic 'User control and freedom'.
- The recognizer has native support for **bimanual interactions**. No ineffective workarounds are needed to combine input for two hands. More about the inherent complexity of bimanual interaction can be found in Section 3.2.7.
- The recognizer understands the data structures that represent our **gestures on a native level**. We can supply a gesture set to our recognizer without having to translate them into concepts that are understood by the recognizer.

The rudimentary mechanics on which our recognizer is built are fairly simple because we can take advantage of the fact that user input is snapped to one of the gesture paths. Our recognizer is neither scale nor rotation invariant, so users are required to draw gestures at a predetermined scale and rotation. Then again, it is very difficult to provide clear mid-path feedback to users about gestures they are drawing if their scale and rotation are unknown until drawn.

3.2.6 Different types of gestures

Our system supports both discrete and continuous gestures. **Discrete gestures** are fairly simple: when the path is fully stroked, the associated action is executed. Continuous gestures have an initializer part and a continuous part. The initializer part is what the user has to stroke before the continuous gesture is activated. What happens after that depends on the type of continuous gesture. We implemented two types: looping gestures and sliding gestures.

The continuous part of each **looping gesture** is a closed path. At each control point, a discrete action is executed. For example, imagine a looping gesture that lets users fast forward through a video. Every time the user reaches a control point of the path, the video skips ahead a few frames. An animated arrow strokes the looping path until a control point is hit for the first time (*Figure 3.15*). It serves as a usage hint (*Section 2.2.7*) to the user.



(A) Initializer part (B) Continuous part

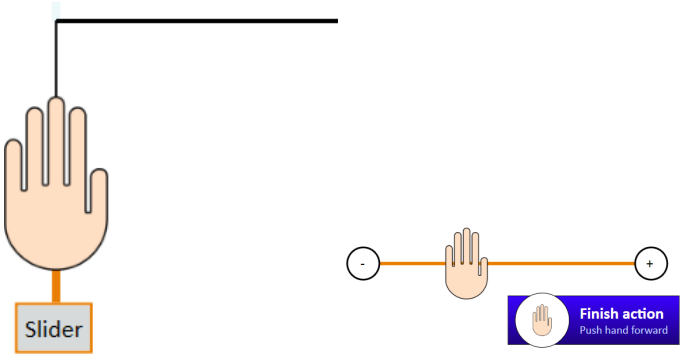
FIGURE 3.15: A looping gesture being executed by the right hand

Later versions of our system improved feedback during looping gestures. An optional icon is rendered in the middle of the looping gesture that plays a press animation whenever a control point is reached (*Figure 3.16*). We also replaced the animated arrow with a static arrow in the middle of the current segment. The arrow fades out as the hand moves towards the segment's end point. Unlike in the first version, the arrow reappears when the user arrives at a new segment.

FIGURE 3.16: Improved feedback during looping gestures

The continuous part of a **sliding gesture** is an open path that represents a value slider. Minus and plus icons are rendered at each end of the path to indicate how the user's position on the path will affect the reported value (*Figure 3.17*). Every position change

is reported to observers so that the program can take an appropriate action based on the current value. Sliding gestures might be appropriate for actions such as zooming.



(A) Initializer part (B) Continuous part

FIGURE 3.17: A sliding gesture being executed by the left hand

Both types of continuous gestures are terminated when the user pushes their hand forward. To communicate this feature to the user, a widget displays a hand whose scale inflates and deflates indefinitely to insinuate a depth push (*Figure 3.18*). The circle behind this hand gradually fills up as the user moves their hand further away. Once filled up, the gesture is completed.

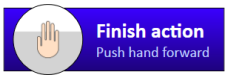
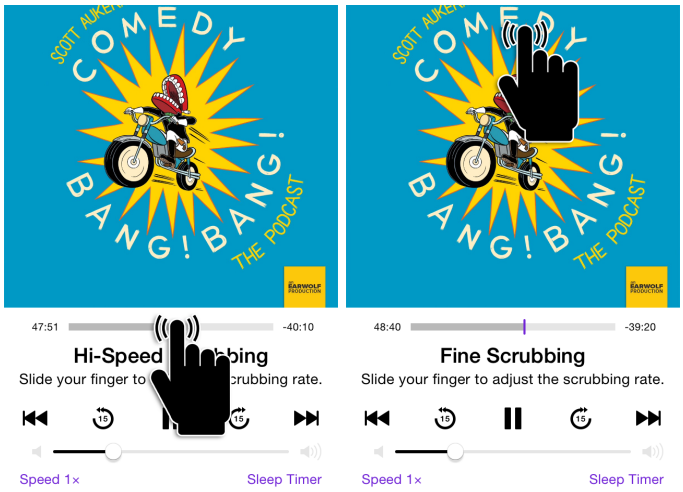


FIGURE 3.18: The finishing tip

Continuous gestures could be expanded to include additional parameters. In some situations it could be helpful to distinguish between slowly and quickly stroked paths. A fast-forwarding gesture could use the looping speed to determine how many frames are skipped at each control point. Another interesting possibility is to take the distance between the raw hand pointer coordinate and the snapped hand pointer coordinate into account. Apple's iOS already implements similar a interaction. When users start scrubbing through audio or video, they can move their finger further away from the playback slider to adjust the granularity of the slider's movement. When their finger is far away from the slider, their finger movements are amplified less to accommodate more fine-grained interactions. An example from the Podcasts app on iOS is shown in Figure 3.19. Such a technique might prove itself to be helpful when the user is trying to adjust the zoom with a sliding gesture.

In later versions we added a feature that gave users the possibility to cancel continuous gestures. If users cancel a gesture before confirming it, any changes that were triggered by this gesture (e.g. volume adjustment using a sliding gesture) would be reverted.



(A) Finger near slider (B) Finger far from slider

FIGURE 3.19: Scrubbing in Apple's Podcasts app with variable granularity

3.2.7 Handling bimanual gestures

Until now, we have focused primarily on unimanual system interactions. When a second hand comes into play, interactions become more complex for a number of reasons.

- Path movements made by one hand can cause gesture candidates to be eliminated and therefore affects which paths are still available to the other hand.
- The cognitive burden on the user is increased because it requires them to process path feedback for two hands simultaneously.
- Since each bimanual gesture consists of two paths, the amount of screen clutter is higher than with unimanual interactions.

As explained in Section 3.2.3, we provide an undo mechanism to restore eliminated candidates so that users can backtrack on their path. Although the concept of the undo stack is relatively straightforward for unimanual interactions, its bimanual implementation is a little trickier. Consider the example in Figure 3.20. The left hand first passes the junction at point 1 and the right hand passes the junction at point 2. Then the left hand passes the junction at point 3 and the right hand passes point 4. If the right hand backtracks past point 4, we need to revert the state of the right hand while preserving the state of the left hand. (If we did not preserve the state of the left hand, its hand pointer would jump to where it was when the right hand passed point 4.) To overcome this issue, we cannot simply pop the top frame from the stack. Instead we will have to perform a merging procedure to ensure that the state of the other hand is preserved and that the list of currently eligible gestures remains in a consistent state.

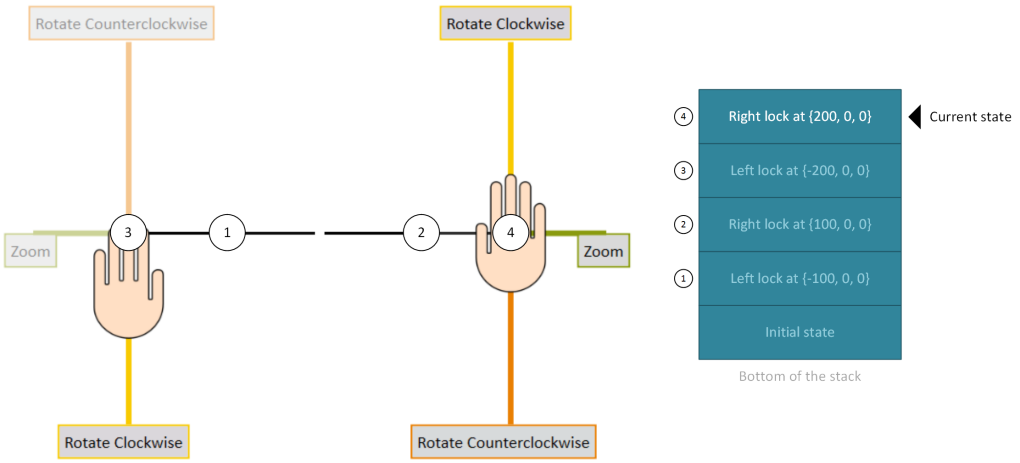


FIGURE 3.20: The bimanual undo stack

A more complex situation emerges if the left hand backtracks past point 3 before the right hand backtracks past point 4. In that case we will not remove the topmost stack frame, but instead remove the last frame in which the left hand passed a junction or bend point. A merging procedure will once again be initiated to construct the new state. As was the case with unimanual gestures, we experimented with discrete and continuous variations. In our system, a bimanual gesture can either hold no continuous parts, a slider path or a looper path for each hand.

3.3 Practical use cases

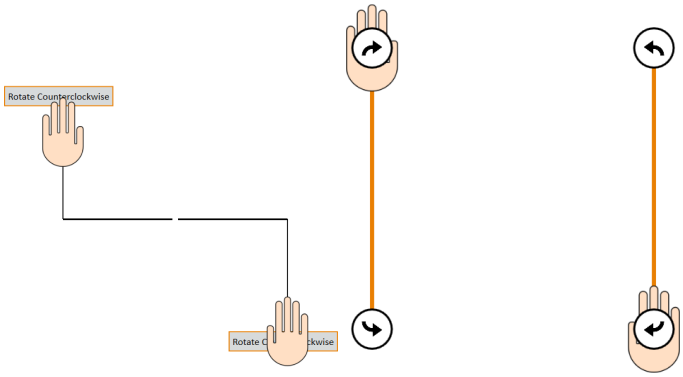
To bridge the gap between our artificial gesture execution sandbox and the real world, we tried to come up with more realistic use cases for our system. First we will provide some practical examples of gestures that our system can support. Then we will describe two simple applications that use our visualization to let users interact with them.

3.3.1 Examples of gestures

We implemented **bimanual zooming** by associating a continuous slider path with each hand. By giving the two individual paths a shared endpoint, it looks like both hands share the same slider. Customized labels at the endpoints of each slider provide a more contextually relevant form of feedforward than generic minus/plus labeling (*Figure 3.21*), which is the default setting.

FIGURE 3.21: A zooming gesture

The **counterclockwise rotation** gesture depicted in Figure 3.22 also uses slider paths for both hands, but in a different way. After the user strokes its initializer part, their hands land on opposite endpoints of separate sliders. By moving their left hand downward and their right hand upward, they can rotate a virtual object counterclockwise. Customized labels on each slider give an additional hint as to how the user's movements will affect the object. An analogous gesture can be implemented for clockwise rotation.



(A) Initializer part (B) Continuous part

FIGURE 3.22: A bimanual rotation gesture

Figure 3.23 depicts a **bimanual looping** gesture. Such a gesture can be useful when the user wants to perform rewind/forward operations on a video. The advantage of a bimanual looping gesture over a unimanual variant is that when the user overshoots their target position (e.g. forwarding a little too enthusiastically), they can compensate by executing the inverse action from within the same gesture. This can be brought back to the Nielsen heuristic 'Help users recognize, diagnose, and recover from errors'. Another interesting thing to point out on Figure 3.23 is that the finishing tip has altered itself to reflect that the user is executing a bimanual gesture.

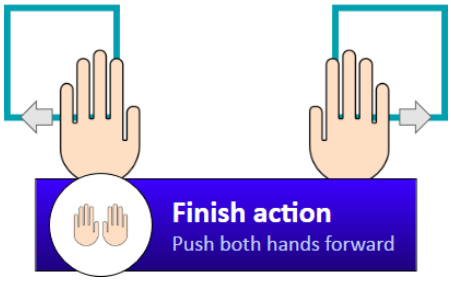


FIGURE 3.23: The continuous part of a bimanual looping gesture

3.3.2 Examples of applications

A video player (*Figure 3.24*)
Initially, a video is shown playing on full screen. Using mid-air gestures, users can execute common playback-related actions such as forward/rewind, play/pause and faster/slower. Whenever users raise at least one of their hands, the video is squeezed into a smaller window so that the visualization does not have to occlude the video. They can then perform gestures as usual. The video automatically regains its full size when users lower their hands or after they finish performing a gesture. We used Big Buck Bunny from the Blender Foundation [6] as our sample video under a Creative Commons license.

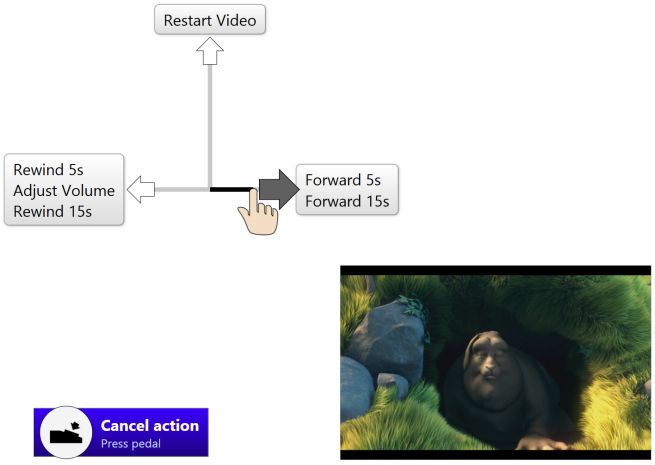


FIGURE 3.24: The video player application during gesture execution

A photo browser (*Figure 3.25*)
Users can navigate through a series of photos with unimanual gestures: first, previous, next and last. Two bimanual interactions provide the user with ways to rotate and scale the current picture. Rotation is performed clockwise or counterclockwise depending on the initializer path that the user strokes. Unlike with the video player, the overlay is rendered on top of the image viewer. To increase the readability of the overlay, a semi-transparent white background is drawn behind it. Because the background is not fully opaque, users can still see what is going on in the application when they are performing a scaling or rotation gesture. All sample images were provided under a CC0 license [12].

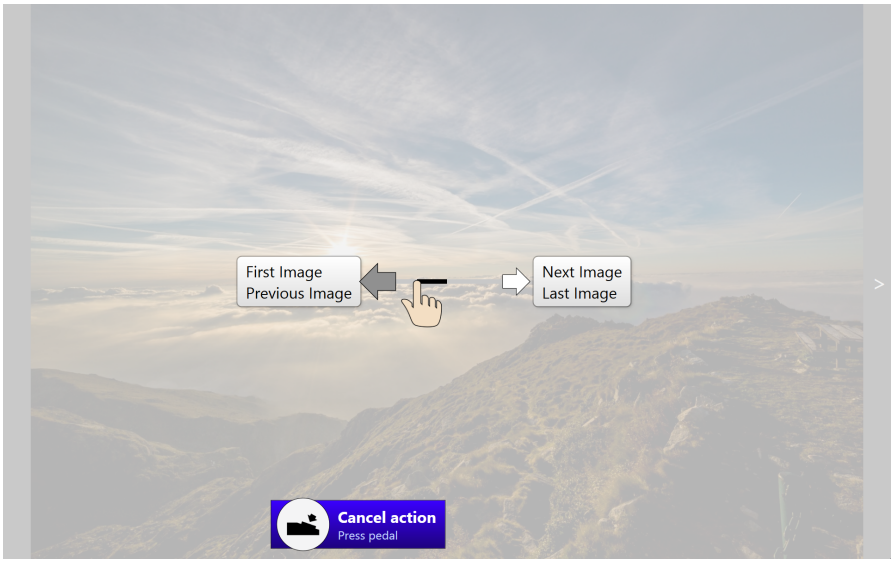


FIGURE 3.25: The photo viewer application during gesture execution

3.4 Conclusion

This chapter described how we prototyped a visualization that guides users through the process of executing stroke-based 3D mid-air gestures. We covered the main features of the visualization, but also devoted time to important backend mechanics that were less visible to users. Our next chapter will describe how we evaluated our prototype to implement further improvements.

Chapter 4

Evaluation of the Mid-Air Gesture Visualizer

Now that our visualization had undergone several iterations of improvements, we planned to subject it to a user test. We were interested to see how well it would assist novice users and which usability issues would pop up. By getting fresh perspectives on the visualization, we hoped to identify most of the remaining issues. Section 4.1 covers our test preparations. We also conducted a pilot study in preparation of the actual study, the results of which are described in Section 4.2. In Section 4.3 we will dive in to our user study and cover the test procedure, our observations and the survey results.

4.1 Preparation

We debated whether we would get more useful results if users tested the visualization in a neutral sandbox environment or within the context of the two applications that were designed specifically for the visualization. The second option more closely reflects how we see our visualization being used in practice and may therefore provide a more realistic experience. In these applications our gestures have actual meaning and when they are executed the state of the application changes. This may provide users with additional feedback (which can improve intelligibility) but may also make unintentional gesture executions more frustrating than they would be in an artificial testing environment. Still, we ended up using a sandbox. If we tested with an application, we would be introducing variables that are not directly related to our visualization. For example, a poorly implemented zooming mechanism could affect a user's subjective satisfaction with regards to the visualization, even though it is not a part of it. One specific concern that came up was that if we were to use bimanual sliders to implement rotation, people

might attempt to make circular movements because of their experience in real-life or with touch-screen applications. By adding the semantics of a rotation gesture to a bimanual slider, we might confuse users because it may not be the most intuitive way to perform rotation. Furthermore, the semantics of our test applications constrained the types and amounts of continuous gestures that we could test.

The gesture set that we used during development was replaced with a more diverse set that aimed to test the limits of the visualization's intelligibility. We included 2 unimanual gestures for every direction[1], amounting to a total of 10 unimanual gestures. We also included 2 bimanual gestures for every available direction[2]. 50% of all gestures incorporated depth movements in their path. We realized that our continuous gestures lacked semantic meaning in our sandbox, which made it hard to test their execution. For this reason, we modified feedback on sliders and loopers. We added a percentage indicator to every slider so that testers could try to confirm at a specific value. To looping gestures we added a counter label that was increased for every control point testers hit. By adding the counter we could ask testers to raise the counter to a specific value and confirm the action. Both types of gestures are shown in Figure 4.1 with their updated feedback.

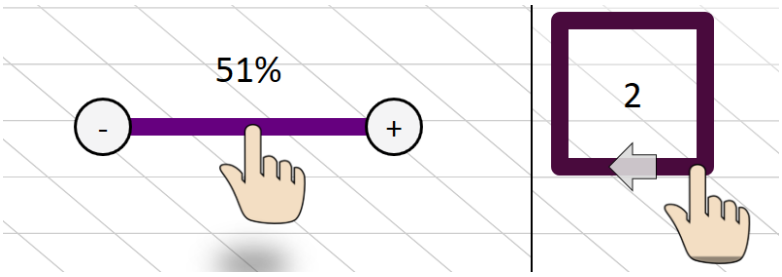


FIGURE 4.1: Updated sandbox feedback on sliders and loopers

We installed a Kinect camera on top of a Dell 2709W monitor to test the visualization. To avoid unnecessary confounding factors in our study, the decision was made to replace the head tilt mechanism with a pedal. It was not our core goal to solve the activation issue that is inherent to mid-air gestures (*Section 3.1*). Since the head tilt had some problems with false positives (wrongful activations) and false negatives (failure to detect activation intent), we opted for the guaranteed accuracy of a pedal. By doing this we avoided that an inaccurate activation mechanism would (a) impede our ability to test the visualization with users and (b) negatively affect subjective satisfaction with the visualization. The pedal was placed at a distance from the monitor that was far enough to satisfy the our Kinect's minimum distance constraints [42].

[1] Up, down, left, right and forward.
[2] Left/right, up, down and forward.

A two-page survey was prepared to collect feedback about their experience with various aspects of the visualization. General demographic information aside, we were interested to know what their dominant hand was, which parts of the visualization they noticed/used and how well the visualization helped them to perform the gesture execution process from start to finish. The full survey as it was presented to test participants can be found in appendix A.

4.2 Pilot study

For our pilot study, we invited two researchers with experience in the field of mid-air gestures to do a test run of the user study and provide additional feedback. Using their input, we identified the need to provide better depth cues to the user. One tester mentioned that a pseudo-3D rendering of a line moving towards the camera looked like it was a thin line with constant thickness and an arrow on top. This effect was unintentional and undermined what was intended to be an important depth cue. We tweaked the visualization in two ways: (1) by increasing the base line thickness and (2) by adjusting the way the line thickness shrinks as the path moves towards the Kinect camera. Figure 4.2 shows the difference in line thickness before the pilot study and before the user tests.

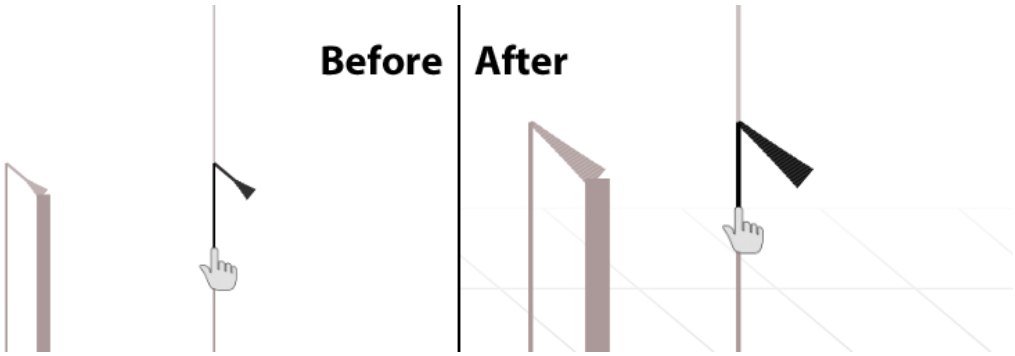


FIGURE 4.2: Line thickness before and after the pilot study

As a result, the visibility of thickness changes in nearby and faraway lines increased. Additionally, we scaled depth coordinates in our 3D projection to make the contrast between forward and backward movements more dramatic. To further improve the overall 3D feeling of the visualization, we designed two additional depth cues. A subtle background that simulated the effect of a 3D cube was added. Its bottom and right side were rendered at low opacity levels to minimize visual clutter. Moving towards the back, the 3D grid gradually faded out to simulate distance. The second new depth cue was a shadow effect for the hand pointer. It was rendered under the hand pointer and near

the bottom of the screen. Y movements were modeled by subtly varying the shadow's opacity. Movements in depth along the Z axis resulted in diagonal shadow movements along the structure of the 3D grid. The 'Cancel action/Confirm action' cues were moved to the left of the screen to minimize shadow occlusion. Figure 4.3 shows how both depth cues were integrated into the visualization. We think that these additions are important in helping people realize that depth movements are not diagonal 2D movements.

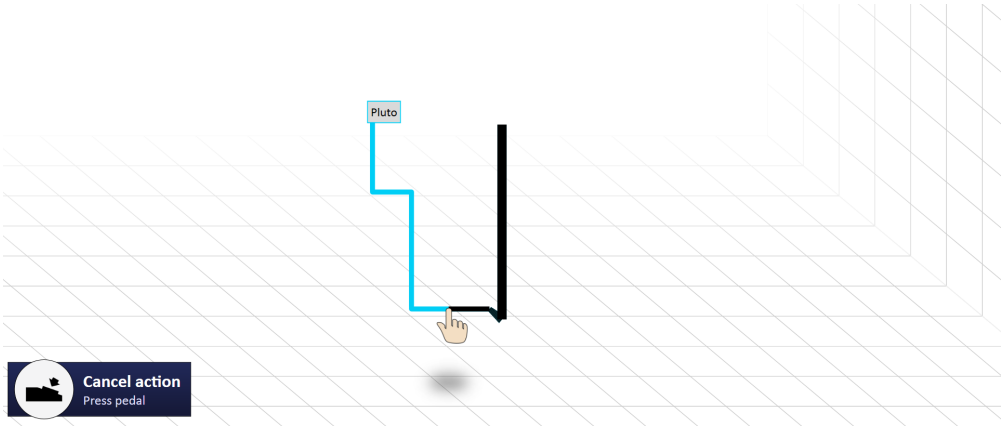


FIGURE 4.3: The cube effect and the pointer shadow, two new depth cues

Based on feedback from the pilot study, we also updated the design of our survey to reduce ambiguity and to increase the granularity of some answer scales. Another interesting point that came up during the pilot was how gestures can become less comfortable depending on the hand position in which the user starts the visualization. Since their hand's position at the start of the gesture is always mapped to origin of the coordinate system, their start position greatly influences how fatiguing the rest of the path will feel. If users start a depth gesture with their arm already stretched, they will have to lean or step forward to complete the gesture. But if they start with their hand near their body, they will have less trouble comfortably executing the same gesture. Following this realization we paid special attention to the posture in which users started gestures.

4.3 User study

This section will explain how we performed our study, document our observations and provide an overview of our survey results.

4.3.1 Procedure

We recruited 14 participants (12 male, 2 female) without any significant knowledge of the gesture visualizer. All of them were either master's students or researchers at Hasselt

University. One outlier was not included in our sample size because we had to halt the test after the participant had significant trouble figuring out how to operate the system. We acknowledge that this result may be indicative of problems with our system, but since we were unable to collect required data from this participant we cannot include them in our statistics.

The user study was performed over the course of two days. Participants were invited to take part in individual 30-minute sessions to test the system. At the introduction of the test, testers were provided with a few essential bits of information. The monitor on which the visualization would later appear only displayed a solid white background during the explanation.

- **Background on the system** - We are testing a system with which users can execute actions using their hands in front of a camera.
- **Background on the test** - We are testing the system, not the user. If anything is unclear or if something goes wrong, do not worry about it. We are only evaluating the system.
- **How to work with the Kinect** - Participants were shown the posture in which they would be using the system: one or two hands held up, with their index finger pointing towards the camera. This helps minimize quality differences in the Kinect's skeletal tracking across participants.
- **How the test will work** - Participants were asked to execute 18 actions using the system. They were told that task descriptions would appear in the bottom right corner of the screen.
- **What to do next** - Additional instructions are provided by the visualization once the test begins.

As shown in Figure 4.4, there were three types of tasks. The leftmost task type was used for discrete gestures. The type in the middle applies to continuous sliding gestures. The rightmost type was used for looping gestures. Each task set included every command from our gesture set exactly once. The target slider/looper values were the same for every participant: 50% for the unimanual slider, 70% + 15% for the bimanual slider, 12 for the unimanual looper and 7 + 30 for the bimanual looper.

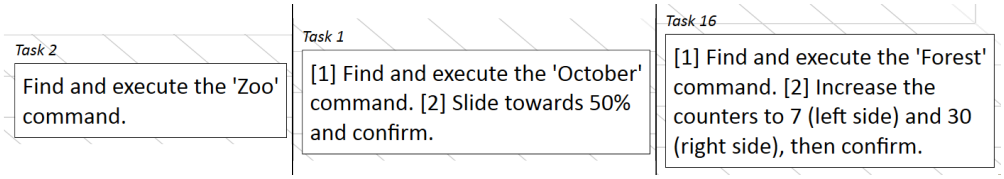


FIGURE 4.4: Task types from left to right: discrete, sliding and looping gesture

Task orders were randomized by generating a random integers on random.org. One generated integer was mapped to each timeslot that was made available to testers. Before each test, we loaded the integer belonging to the current timeslot from a text file. This integer was then used as a seed for our internal task randomizer. Since the same seed always results in the same task order, we could always resume the test even if the program crashed. We believe that the randomized task better reflects the learning process of real-life usage, given that there is no fixed order in which new users perform their first commands.

During test execution, various pieces of information such as gesture execution speeds, gesture cancellations and hand (de-)activation events were written to disk. The information was saved in a human-readable plaintext format as well as a binary format. In addition, we logged every raw input point so that we could replay any user's hand movements after the conclusion of the study if needed.

After the test, we asked participants to fill in a two-page survey consisting mostly of closed questions. A general comment field was also included just in case testers wanted to address any additional topics that were not sufficiently covered by the questionnaire. After participants filled in the survey, we asked additional questions based on what had happened during the test.

4.3.2 Observations

While participants were working their way through their task set, we wrote down any interesting observations that came to mind. We asked additional questions to our testers to verify and complement our observations. Although we cannot discuss everything we wrote down, we will use this section to cover a few major reoccurring observations.

- **Unintentional mapping of gesture overviews to left/right hand** – Some participants were confused by the placement of our gesture overviews (*Figure 4.5*). As soon as users held up any number of hands, the system showed one gesture overview at full opacity and the other overview at low opacity[3]. But sometimes they would mistakenly assume that the overviews were mapped to the left and right hand, despite the 'One-handed actions' title at the top of the overview. The opacity effect did seem to help testers in figuring out the meaning of the gesture overview. One obvious way to avoid the expectation of left/right hand mapping

[3]When the user held up one hand, the 'One-handed actions' overview was shown at full opacity and the 'Two-handed actions' overview was shown at low opacity. The opposite applies to bimanual interaction.

is to place the overviews at the bottom and top of the screen, but due to limited screen size we would have to split the gesture overview into multiple columns.

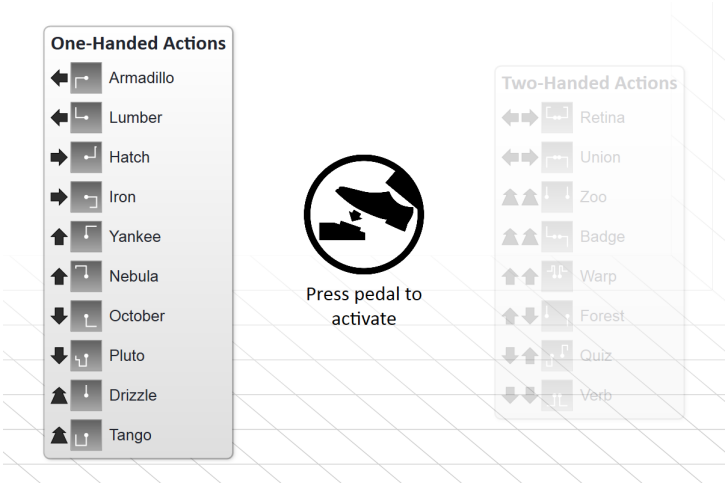


FIGURE 4.5: Gesture overviews with the opacity effect

- **Path previews in the gesture overview were confusing** – In talking to our testers, it seemed like these path previews did more harm than good. One tester mentioned that they thought they had to memorize this information and use it after they pressed the pedal, which unnecessarily increases the cognitive burden of the visualization. In reality the information in the path preview is redundant, so users do not need to memorize it. Several testers mistook the meaning of the preview. They thought that the dot in the middle was the end point of the gesture rather than the start. By accidentally setting up the wrong expectation beforehand, we may diminish their feeling of being in control. Some interpreted the dot as an arrow, which only strengthens the effect. For one gesture whose last line segment was rather small, the last segment was hardly visible in its preview. Another issue that we had anticipated on but did not really came up was that the preview could not convey depth cues. Due to its limited size, we had to use a parallel projection from 3D to 2D. It is not unlikely that some of the issues with path previews disappear on larger screens, e.g. when a wall projection is used. But given that the previews are not reaching their goal of helping users (and appear to be doing quite the opposite), we suggest that they are removed from the visualization.

- **People's starting position affected fatigue** – Although participants were shown that they needed to point their index finger(s) towards the camera, we

did not instruct them to hold their hands at a certain distance from their body[4]. As a result, we saw differences in the way each participant held their hands. People who started with their hands further away from their body had a tougher time performing certain movements, especially when depth was involved. This happened because they arrived at the end of their comfortable gesturing range more quickly than participants who started with their hands held close to their body. One tester reported feeling fatigue around task 8 of 18. We suggested that they started the visualization with their hands held closer.

The influence of participants' starting positions became most apparent when they had to confirm gestures. Section 4.2 already mentioned that it is harder for people to move the same distance towards the camera when their hands are already stretched than when their hands are still closer to their body. One obvious solution would be to shorten the confirmation distance to make the movement comfortable in both cases, but then we would get more accidental confirmations. We also want to avoid telling users to start in a certain position, since we want to encourage them to do what feels most comfortable. Further thinking needs to be done on this issue to find a better solution.

- **Simulated 3D often went unnoticed at first** – In spite of our efforts to introduce additional depth cues, participants still had trouble discovering how 3D gestures worked. Initially we had three depth cues: the scale of the hand pointer (smaller is further away), the thickness of lines (smaller is further away) and the cabinet projection effect (movements parallel to diagonal lines indicate distance changes). Following our pilot study, we added a "cube" background, implemented a shadow for the hand pointer, tweaked the line thickness effect (*Figure 4.2*) and visually stretched out Z coordinates to make depth movements more pronounced. We often asked participants which depth cues they noticed and many indicated that they did not consciously see the shadow and line thickness effects. The background was a little more noticeable, but not everyone made the connection to 3D gestures. When participants were asked to perform a gesture that immediately moved towards the Kinect, the 'forward' arrow would often clue them in. But when they were asked to perform a gesture that included depth movement later in its path, they were more likely to struggle if they had not executed a 3D movement before. We believe that depth cues will be more successful if they focus on providing depth information *before* the user moves in depth (e.g. line thickness effect) rather than *after* they do so (e.g. moving shadow). Failure to properly

[4]We implemented our visualization so that a user's hand position at the start of the gesture would always be mapped onto the hand pointer's starting position, so it did not matter how they held their hands. After the test we asked some participants whether they were aware of this, and most of them were not.

communicate depth prompted users to try diagonal movements when they were supposed to move in depth. In some cases we had to tell participants to try moving towards the camera.

- **Sometimes the 'forward' arrow in the starting visualization was misinterpreted** – The placement of the 'forward' arrow (*Figure 3.7*) turned out to be a little unfortunate. Sometimes people would take it to mean that they first had to move left from the center and then had to push forward when they were near the arrow. Our suggested fix is relatively easy to implement: move the arrow so that it sits on top of the forward-moving line.

- **Clutching** – The term 'clutching' usually describes situations where a user lifts and repositions their mouse on the table to reach a far-away target [52]. We see a similar phenomenon happening with our system. Sometimes testers would get stuck and restart the visualization. This can happen for multiple reasons. They may find themselves struggling to perform a movement because their hands are already fully stretched. This may prompt them to re-establish the mapping between physical and virtual space by restarting. Another possible reason is that their pointer may have ended up in the wrong part of the visualization and they found it easier to restart than move back their pointer. While these issues are not directly related to our visualization, they do affect the user experience.

- **Discovery of the 'Cancel/Confirm' mechanism** – On multiple occasions, a tester would accidentally cancel a continuous gesture instead of confirming it when they were first faced with one. Once they realized what had gone wrong they performed the gesture again. One tester suggested that they may have overlooked the 'Cancel/Confirm' cue because it was tucked away in a corner. They also remarked that looking at the hints could cause accidental movement on their path, so it may be helpful to move the hints closer to the pointer. Then again, we do think that pointer-following cues will increase clutter and occlusion. Furthermore experienced users might get irritated if the hints are displayed too prominently.

- **Synchronous vs. sequential bimanual interactions** – There are three ways in which bimanual interactions can be performed: sequentially (first left path then right path, or vice versa), synchronously or a combination of the two. We did not tell participants how they were supposed to execute bimanual gestures, but our system can accommodate all three strategies. Each of them was used by at least one person, so it is a good thing that our system provides this degree of flexibility.

- **Participants used coarser but faster loop movements for longer tasks** - We included a looping gesture in our test that demanded a little more physical

movement to complete. Our main reason for including a longer looping task was to test whether participants would try to speed up the movement. We were concerned that the rectangular appearance of the loop would give the impression that the user's movements had to be perfectly rectangle-shaped. Although we designed the recognizer to cope with rough ellipse-shaped movements that were easier to perform than rectangle-shaped movements, we feared that users would not catch on to this possibility. In reality, it seemed like most participants did in fact attempt to use coarse movements to speed up the process.

- **Participants unsuccessfully tried to compensate for overshoots in loop movements** - When we originally implemented looper movements, we blocked backward movements beyond the most recently hit bend point. By preventing unintentional backward movement, we wanted to make it easier for users to complete looping tasks with coarse movements. If an undo mechanism made semantic sense (e.g. in a rewind/forward gesture for a video player), we planned to provide a second loop with which the opposite effect could be achieved (*Figure 3.16*). This also seemed easier to explain to users than if we were to integrate two opposite actions in the same loop.

  However, during our user test we experienced the drawbacks of this decision. Due to the artificial nature of our testing environment and the structure of the counter manipulation task, it made sense for people to try the inverse movement. This happened even though the visualization did not provide any feedback explaining the semantics of this movement. Based on this observation, we would consider removing the constraint for certain tasks.

- **Confirmation of continuous gestures lead to accidental movement** – This is an effect that we had anticipated on, but our solutions have so far been task-specific. In the photo browser from Section 3.3, we implemented rotation as a bimanual slider movement whose input values are in many cases snapped so that the picture is rotated to multiples of 90 degrees. A more structural solution could, for instance, involve users changing the posture of their hands to confirm their action. Because of limitations to our current camera hardware, this was not yet an option. One tester tried to confirm a unimanual continuous gesture with their other hand to prevent accidental movement on their active hand, but such functionality had not been implemented.

- **Accidental gesture activations** – Sometimes participants would accidentally trigger discrete gestures against their will. We were unable to identify the exact cause. It could be due to any number of causes from this non-exhaustive list: Kinect tracking inaccuracies, unintentional physical movements, glitches in our

recognizer or unawareness of how their physical movements were being interpreted. To address accidental activations, we would need to do additional tests so that we can narrow down what the exact problems are.

- **Some people moved their head forward to see the screen better** – When we asked these participants whether they could see everything on the screen, they did not report having any readability issues. That said, we do think that participants would have an easier time using the visualization on a bigger screen because (a) depth cues and text would be bigger and (b) it would be better from an ergonomic perspective if people did not feel a need to move their head forward.

4.3.3 Survey results

In the first section of the survey, we asked our participants for some general demographic information. As Figure 4.6 shows, most of our testers were 18-24 years old, identified themselves as male and used their right hand to write. There were no ambidextrous participants in our study.

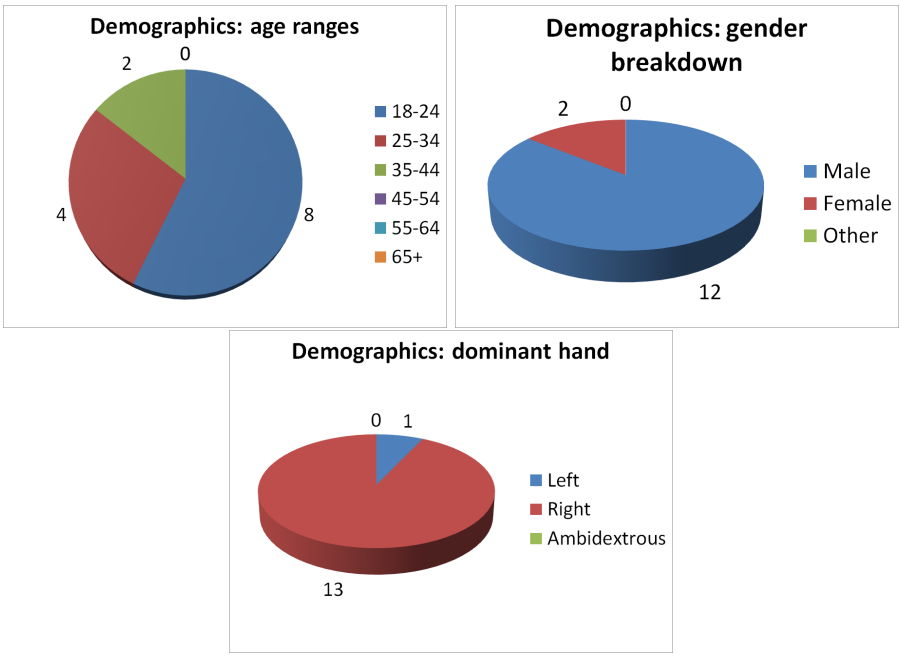
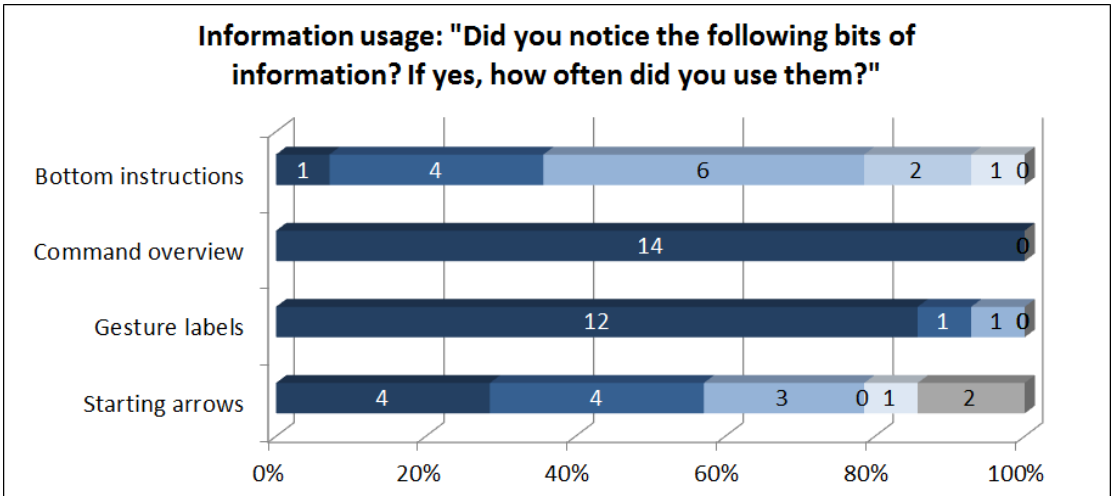


FIGURE 4.6: Demographic information about our participants

The second part of the survey focused on the visualization. We were interested to see which bits of info were noticed by people and how often they were used. The respondents' answers to this question are summarized in Figure 4.7. The command overview that pops up when a user raises any number of hands was used most heavily, with 100% indicating that they always used it. The gesture labels next to the command paths

came in a close second with 86%. Participants reported using the arrows at the start of the visualization a lot less and 2 people (14%) answered that they did not notice them. The cancel/confirm instructions at the bottom of the screen were used less than the starting arrows, but they were noticed by everyone. The survey included a screenshot of each feature to ensure that the respondents knew what we were referring to.



	Starting arrows	Gesture labels	Command overview	Bottom instructions
Yes, always	4	12	14	1
Yes, often	4	1	0	4
Yes, sometimes	3	1	0	6
Yes, rarely	0	0	0	2
Yes, never	1	0	0	1
No, did not notice	2	0	0	0

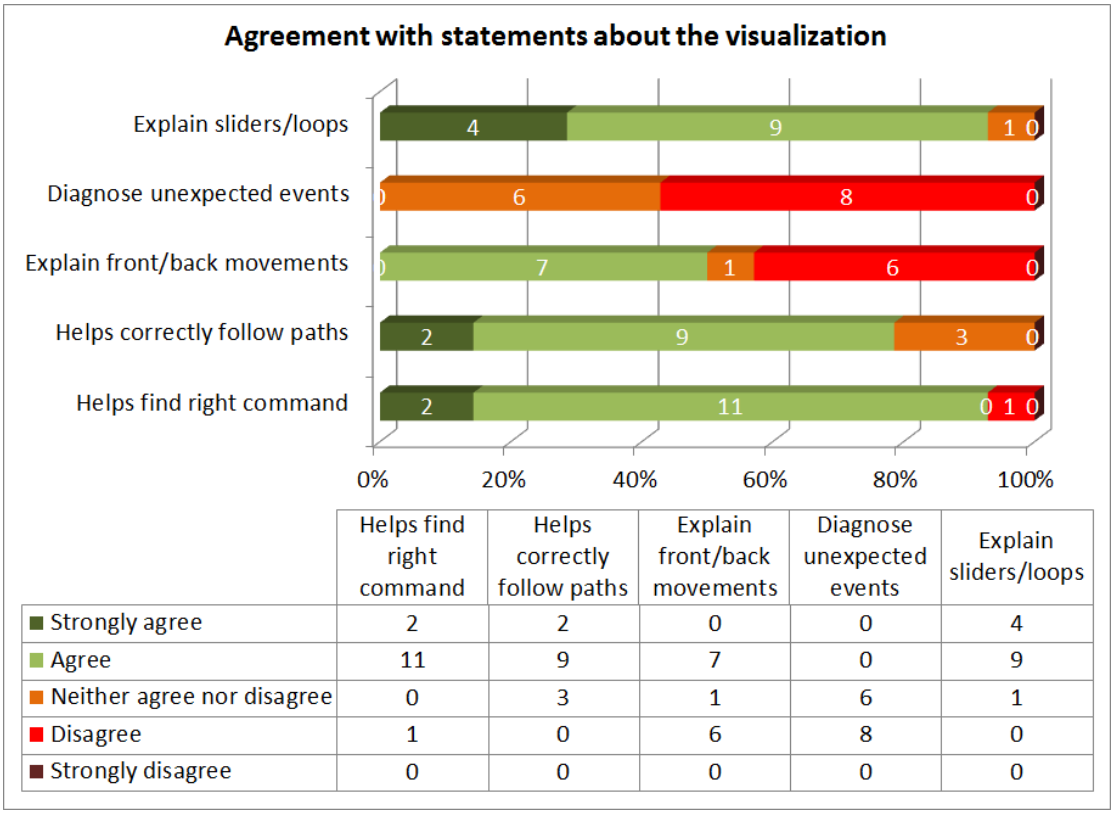
FIGURE 4.7: Usage statistics as reported by the participants

We then asked participants to indicate their level of (dis-)agreement with the following statements on a five-point Likert scale:

1. The system helps me to easily find the right command from a list of possibilities
2. The system helps me to correctly follow a command's path
3. The system clearly shows when a gesture requires forward/backward hand movements
4. The system helps me to figure out what went wrong when something unexpected happens
5. The system helps me to understand how slider and loop commands work

The last statement was accompanied by a screenshot for clarity. Figure 4.8 summarizes testers' responses to this question. The statement about slider and loop commands got

the most 'strongly agree' votes. 93% ticked 'strongly agree' or 'agree' for statements 5 (sliders/loopers) and 1 (finding the right command), but less people strongly agreed with the latter. It is apparent that 3D movements were less clear, given that 43% indicated some form of disagreement with statement 3. Statement 4 (diagnosing unexpected events) was never agreed with and saw 57% disagreement, which indicates that more work needs to be done to improve our score for the Nielsen heuristics 'Error prevention' and 'Help users recognize, diagnose, and recover from errors' [60].



	Helps find right command	Helps correctly follow paths	Explain front/back movements	Diagnose unexpected events	Explain sliders/loops
Strongly agree	2	2	0	0	4
Agree	11	9	7	0	9
Neither agree nor disagree	0	3	1	6	1
Disagree	1	0	6	8	0
Strongly disagree	0	0	0	0	0

FIGURE 4.8: Overview of statement ratings by the participants

4.4 Conclusion

In this chapter we described how we evaluated our system in two ways. First we invited two researchers to take part in a pilot study of our system. Using our observations and their feedback we implemented further improvements, most of which had to do with depth cues. Then we organized a user study to further assess the intelligibility of our system. 14 out of 15 participants were able to execute a complete set of 18 gestures using our visualization. 93% of all participants who completed the test agreed or strongly agreed that (a) the system helped them find the gestures they were looking for and that (b) it helped them understand how our continuous gestures worked.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Our goal for this thesis was to implement a visualization that could guide users through the process of executing 3D mid-air gestures. Before we started developing the system, we surveyed a broad range of various existing techniques that facilitate the user's UI learning process. Based on the differences and similarities between all the examples that we collected, we defined 13 categories of UI learning techniques. In our discussion of each category, we elaborated on their characteristics and provided specific examples from existing software, games and/or academic research. Category comparison tables at the end of the chapter further examined their individual differences. Special attention was paid to gesture learning systems because they were most relevant to our later work.

After our initial literature study, we started working on our second and most important contribution: the stroke-based 3D mid-air gesture guide. Our work explains the most important aspects of the visualization, varying from user interaction elements to backend mechanics that support the gesture execution process. The system supports both unimanual and bimanual gestures. In addition to traditional discrete gestures, we developed two types of continuous gestures: sliders and loopers. Practical use cases were provided to show how our gestures could be incorporated into real-world applications.

Following the development of our prototype, we subjected the system to a pilot study that informed further refinements to the system. Then we organized a more formal user study to observe how people would interact with the system. We documented major reoccurring observations and discussed the results from our survey. During our test, 14 out of 15 participants were able to successfully execute a set of 18 gestures using the visualization. The set was carefully constructed to include 2D/3D gestures,

discrete/continuous gestures and unimanual/bimanual gestures. 93% of participants who completed the test agreed or strongly agreed that (a) the system helped them find the gestures they were looking for and that (b) it helped them understand how our continuous gestures worked. Some participants had problems perceiving our depth cues, so more research will have to be done to improve them.

5.2 Future work

In this section, we list additional ideas that could either improve existing parts of the visualization or expand the scope of our work into new territory.

5.2.1 Better depth cues

During our user tests, it became apparent that people still had trouble perceiving our environment as simulated 3D. We suggest two changes that might improve the 3D-ness of the visualization. By stretching Z coordinates, depth differences would become more pronounced. This could help in situations where some testers had trouble perceiving the depth effect at a distance, like on the junction where Pluto and October diverged (*Figure 5.1*). Secondly, we think that small arrows at junctions could help raise awareness of depth movements. By adding arrows to each junction like we did in Figure 3.7, we hope to make movement possibilities more obvious. To avoid big increases in clutter, the arrows would only be shown on junctions when the hand pointer is nearby. These are of course just ideas and they will need to be tested with users to judge their effectiveness.

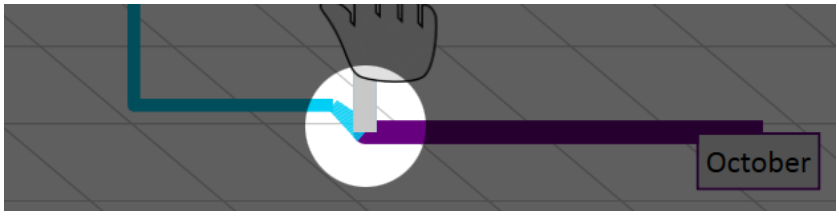


FIGURE 5.1: A less pronounced depth difference in our visualization

5.2.2 Support for more complex gesture paths

Our system implements gestures as combinations of horizontal and vertical lines. This is a limitation to our implementation, not our concept. We believe that we struck a good balance between extending our recognizer to support expressive gesture sets on one hand and focusing on the core goals of this thesis on the other hand. Future work could add support for more complex gestures that incorporate arc-shaped movements,

but research by Nancel et al. [58] suggests that linear gestures are generally faster and more often preferred than circular ones.

The gestures in our system always use angles that are multiples of 90 degrees. As a result, our visualization looks less cluttered than it would if the starting visualization in Figure 3.7 had to display additional directions. We also want to avoid unnecessary confusion between diagonal movements and depth movements. In our tests we already observed that some people tried moving diagonally when they had to move towards the camera, even though none of our gestures incorporated any diagonal movements. Allowing arbitrary angles would arguably serve to increase confusion.

5.2.3 Support for posture changes

ShadowGuides [25] plays with the idea of having users change the posture of their hand during multitouch interactions. We see a way to integrate similar posture changes into our system. Although our Kinect camera had limited support for various hand postures [43], it could not distinguish between them with satisfactory accuracy when we tried it out. We also found that the hand's posture affects the accuracy of the Kinect's skeletal data. In our experience, the camera did a better job of tracking the hand's position when we were using a pointing posture than when we were using a stretched hand posture. This limitation forms an additional technical hurdle to the implementation of posture changes.

We have created some mockups to illustrate how we would implement posture changes during gesture execution. Each of the segments that make up a gesture is associated with a posture. By default, it is the pointing posture. If the posture needed to stroke the next segment is different from the posture that the user is currently using, the rest of the path is locked away behind a "posture gate". Their look is similar to arrows in the starting visualization (*Figure 3.7*). The mockup in Figure 5.2 shows the user stroking the path with a pointing posture until they reach a posture gate. They will not be able to pass this gate unless they change their posture. Notice that they can execute 'Looper' without changing their posture because it uses the default posture on its entire path.

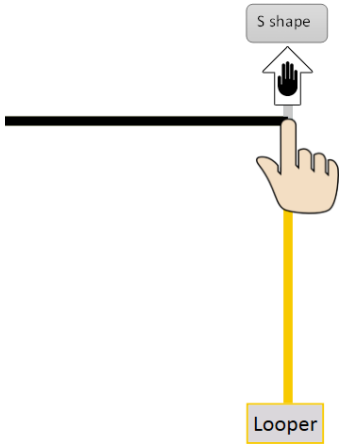


FIGURE 5.2: Before posture change

When users do change their posture, several things will happen. The rest of 'S shape' will reveal itself, the hand pointer will assume the new posture and the posture gate will disappear. This situation is mocked up in Figure 5.3. It is fully reversible in case users decide to take a different path for any given reason. Since 'Looper' uses the default posture on its entire path, a posture gate is set up at the start of its next segment forcing users to change back to a pointing posture. Users may be required to assume their previous hand posture if they want to backtrack on their path, but we do not enforce this in our mockup because it decreases convenience.

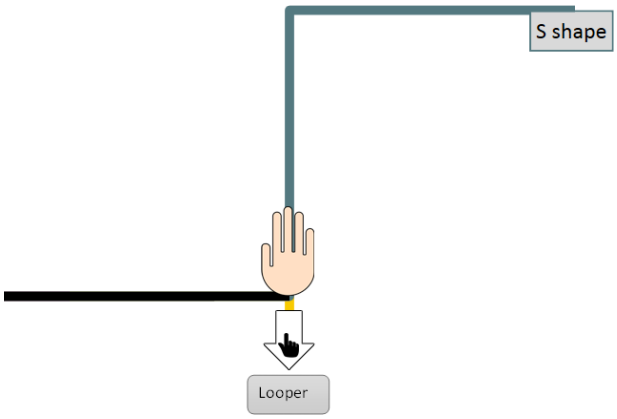


FIGURE 5.3: After posture change

Although we can imagine that posture changes could help attach semantics to seemingly arbitrary gesture paths in some cases, it is likely that they will slow down gesture execution and increase the cognitive load associated with the visualization. Users might be perfectly content stroking simple but meaningless gesture paths if their execution times are faster. We expect that the appeal of posture changes will be limited, but there might be uses for them that we have not yet uncovered.

5.2.4 Tool support for developers

Using a special gesture set creator, developers could design custom gesture sets for their programs. Evaluation heuristics could be built in to the editor so that developers can optimize usability aspects of their gesture set. In 2014, Hincapié-Ramos et al. developed the *Consumed Endurance* model to quantify arm fatigue in mid-air interactions [31]. Long Jr. implemented a gesture editor for pen-based interfaces that tracks each gesture's similarity to other gestures in the set [50]. Such measurements could help developers to make gestures more distinguishable, which may increase their memorability.

5.2.5 Ergonomic improvements

This item goes back to a problem that was first mentioned in Section 4.3.2. Depending on the posture in which participants started their gestures, some movements would become rather tiring to perform. We considered scaling down the size of physical depth movements to reduce fatigue effects, but then we would increase the risk that the hand pointer starts making unintentional depth movements and that we record accidental slider/looper confirmations. Another possibility is that users can be cued to start with their hands held closer to their body. The downside to that solution is that it would force users to start gestures in a way that may not feel the most comfortable to them. So far we have not been able to identify an ideal solution, but we still believe that there are opportunities to improve the overall ergonomics of the system.

5.2.6 Adapting the visualization to multiuser environments

From a technical standpoint, it is feasible to extend our visualization to multiple users. Most of our software architecture was written in a way that would allow expansion into multiuser environments later on, but we did not actively explore that path in our implementation. We did cover some of the general design challenges that revolve around collaborative multiuser environments in Section 2.4.

When multiple people are executing gestures simultaneously, we envision each person working with their own separate visualization. The horizontal position of the visualization will be linked to the user's position so that it is rendered where they are standing. This will not work in every possible scenario. When there is opportunity for confusion about which visualization belongs to which person, avatars could be rendered on the screen to take away the ambiguity. Facial recognition features in the Kinect SDK can help the system associate each user with the right avatar.

In some cases, commands issued by multiple users can conflict with one another or produce unintended consequences. To avoid conflicts, our visualization could take measures to make users aware of each other's intentions. A more foolproof but restrictive alternative would be to artificially limit the system to one simultaneous user. Then again, a one-user limit would make it impossible to implement cooperative gestures like the ones Morris et al. [57] described. For this reason, floor control might prove itself to be a more suitable conflict avoidance mechanism. When floor control is implemented, actions impacting multiple users would need to be confirmed by every impacted user - by holding their thumbs up, for instance.

Appendix A

The User Survey

After this page, we have included a blank copy of the survey that was presented to every participant in the user study that was described in Section 4.3.

...../05/2014 ..... :.....

Date and time of the test

General information

What is your age range?

☐ 18-24	☐ 25-34	☐ 35-44
☐ 45-54	☐ 55-64	☐ 65+

What is your gender?

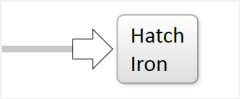
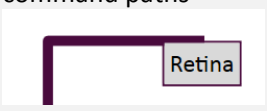
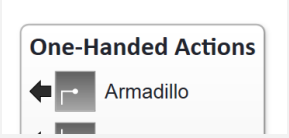
☐ Male	☐ Female	☐ Other

What is your best writing hand?

☐ My left hand	☐ My right hand	☐ No preference

About the system

Did you notice the following information? If yes, how often did you use it?

	Noticed?	If yes, how often did you use it?				
		Always	Often	Sometimes	Rarely	Never
The arrows at the start of the visualization 	Yes / No					
The text labels next to the command paths 	Yes / No					
The command overview that pops up when you raise one or two hand(s) 	Yes / No					

	Noticed?	If yes, how often did you use it?				
		Always	Often	Sometimes	Rarely	Never
The instructions at the bottom of the screen 	Yes / No					

Rate the following statements.

	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
The system helps me to easily find the right command from a list of possibilities					
The system helps me to correctly follow a command's path					
The system clearly shows when a gesture requires forward/backward hand movements					
The system helps me to figure out what went wrong when something unexpected happens					
The system helps me to understand how slider commands *(left image)* and loop commands *(right image)* work 					

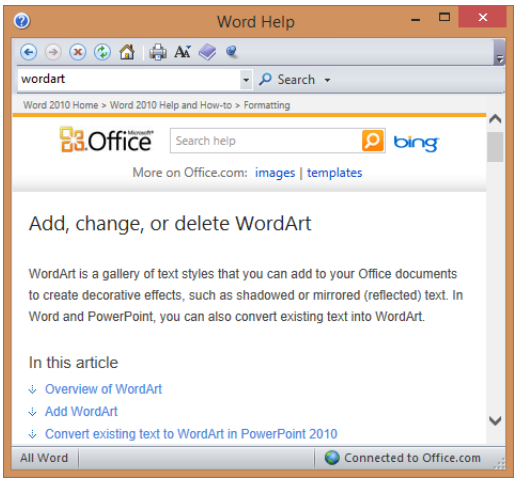
If you have any other comments or thoughts about the system, please write them below.

Appendix B

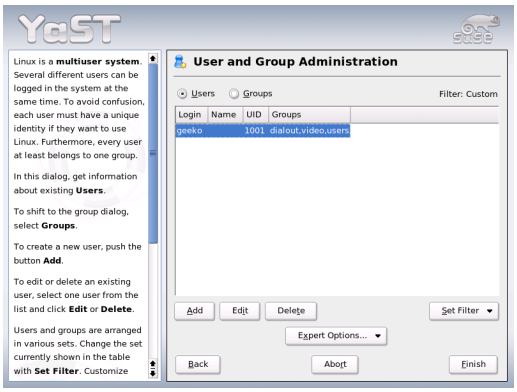
Dutch Thesis Summary

Het is nog niet zo lang geleden dat computergebruikers eerst een dikke handleiding moesten lezen voordat ze aan de slag konden met hun nieuwe aankoop, maar in de praktijk staan weinig mensen te popelen om eerst een droge pil van honderden bladzijden achterover te slaan. Dit fenomeen wordt door de literatuur onderbouwd en kreeg de naam *production bias* mee [11]. Gebruikers willen hun werk gedaan krijgen, maar zijn over het algemeen niet gemotiveerd om het systeem goed te leren kennen of om uit te zoeken hoe ze op de lange termijn efficiënter kunnen werken. Handleidingen zijn dus niet ideaal en daarom zouden leermethoden die rechtstreeks op hun doelen aansluiten waarschijnlijk meer geapprecieerd worden.

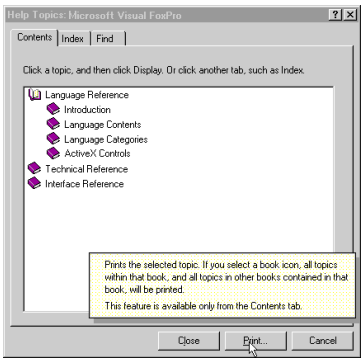
Later maakten hardwareverbeteringen de opkomst van grafische gebruikersinterfaces mogelijk. Dankzij deze evoluties kregen softwareontwikkelaars en onderzoekers meer vrijheid om te experimenteren met betere alternatieven voor handleidingen. Ze begonnen op allerlei manieren instructies te verwerken in de gebruikersinterfaces zelf. Zo werd het voor gebruikers eenvoudiger om onbekende gebruikersinterfaces te leren kennen. In het eerste deel van deze thesis hebben we op basis van academische en gecommercialiseerde voorbeelden alle vormen van UI-leerhulp die we verzameld hadden onderverdeeld in 13 categorieën. Figuur B.1 en B.2 tonen voorbeelden uit elke categorie.



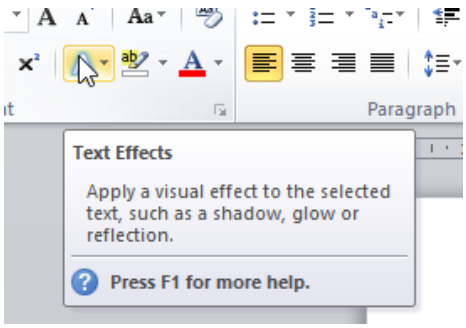
(A) Standalonedocumentatie



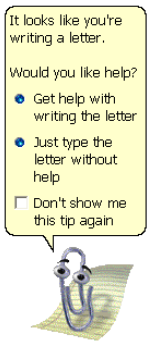
(B) Geïntegreerde hulp in SuSE Linux (bron: [77])



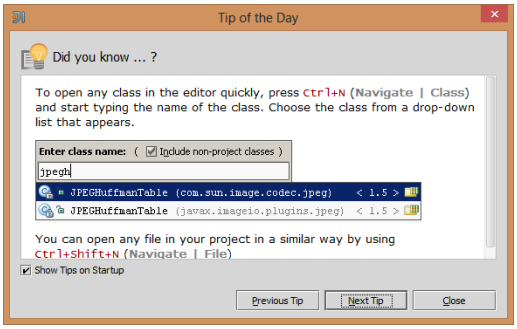
(C) Een "Wat is dit?"-popup in Visual FoxPro (bron: [35]))



(D) Een tooltip uit Word 2010

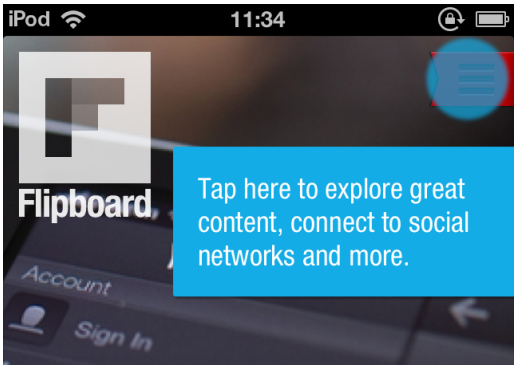


(E) Embodied agent in Office (bron: [23])



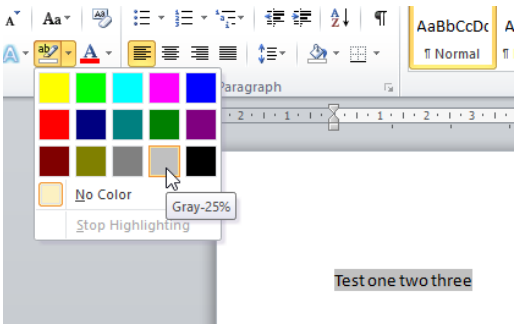
(F) Niet-doelgerichte tips in IntelliJ IDEA

(G) Gebruikshint in Marble Blast Gold



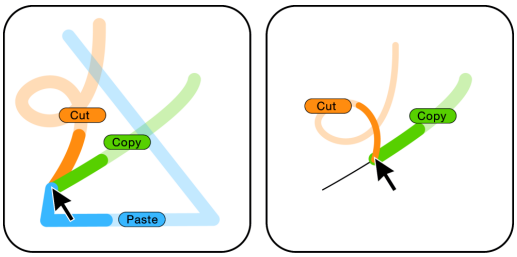
(H) Functieintroductie in Flipboard

FIGUUR B.1: De eerste acht categorieën van UI-leerhulp in onze classificatie

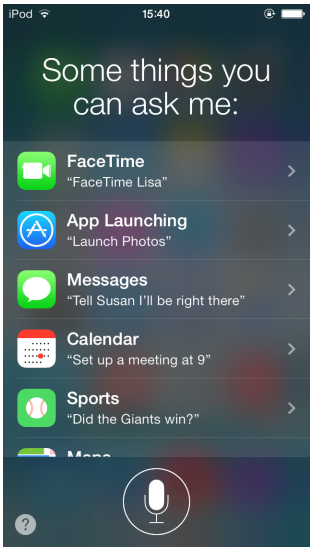


(A) Actiepreview in Word 2010

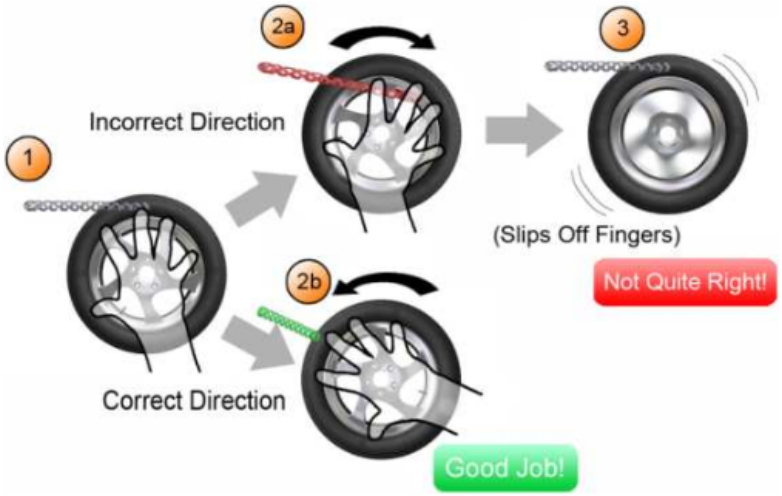
(B) "En nu?"-hulp in Civilization V



(C) Gebaarleersysteem van Bau en Mackay (bron: [5])



(D) Spraakleersysteem in iOS 7



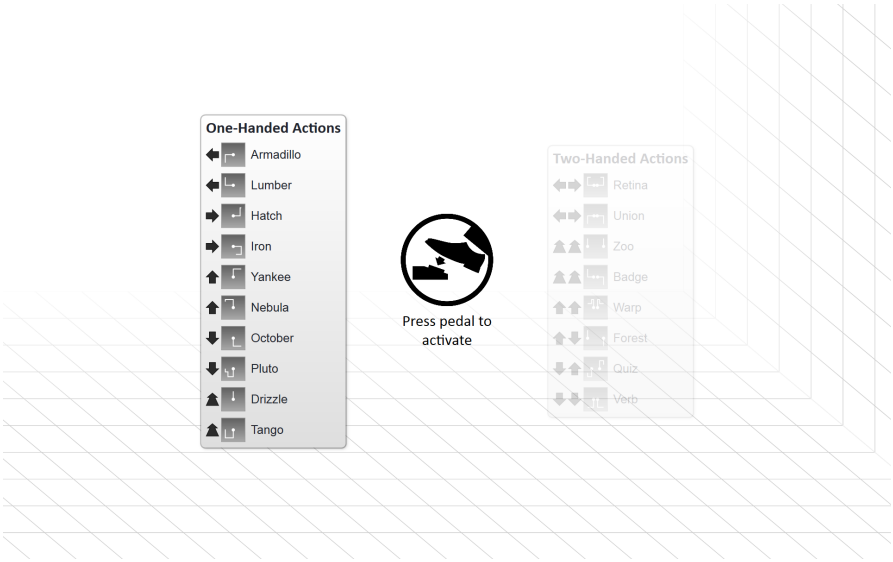
(E) Trainingstaken in Gesture Play (bron: [8])

FIGUUR B.2: De laatste 5 categorieën van UI-leerhulp in onze classificatie

Na afronding van onze studie over bestaand werk rond UI-leertechnieken begonnen we onze hoofdcontributie te ontwikkelen: een visualisatie die gebruikers helpt bij de uitvoering van stroke-based 3D mid-air gestures voor een Microsoft Kinect-camera. Met

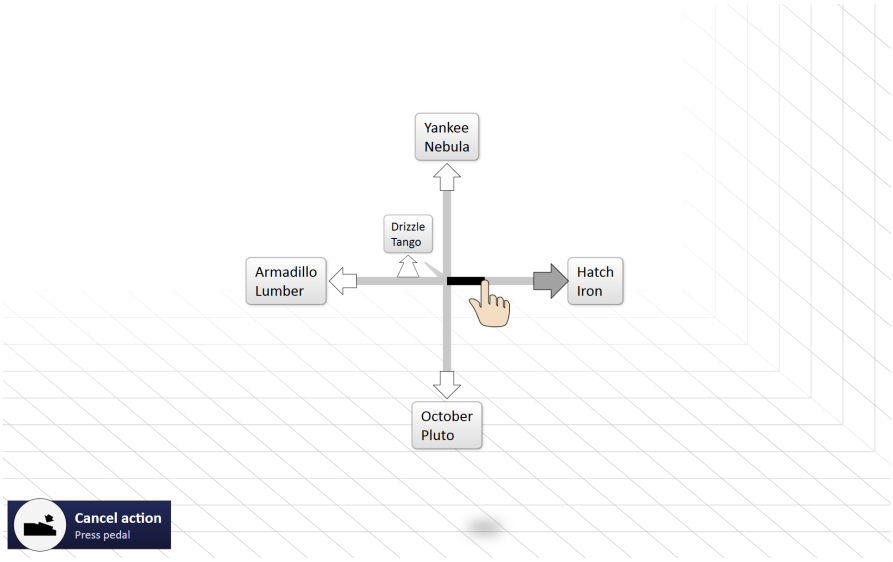
dit systeem krijgen gebruikers voortdurend feedback terwijl ze 3D-gebaren tekenen in de lucht. Door expliciet feedback te geven tijdens het gesticuleerproces zien gebruikers wat ze doen en kunnen ze hun bewegingen voortdurend bijsturen.

Het gesticuleerproces werkt als volgt. Gebruikers steken eerst één of twee handen omhoog. De visualisatie vertelt hen vervolgens welke commando's ze met deze hand(en) kunnen uitvoeren, zoals ook geïllustreerd wordt in Figuur B.3.



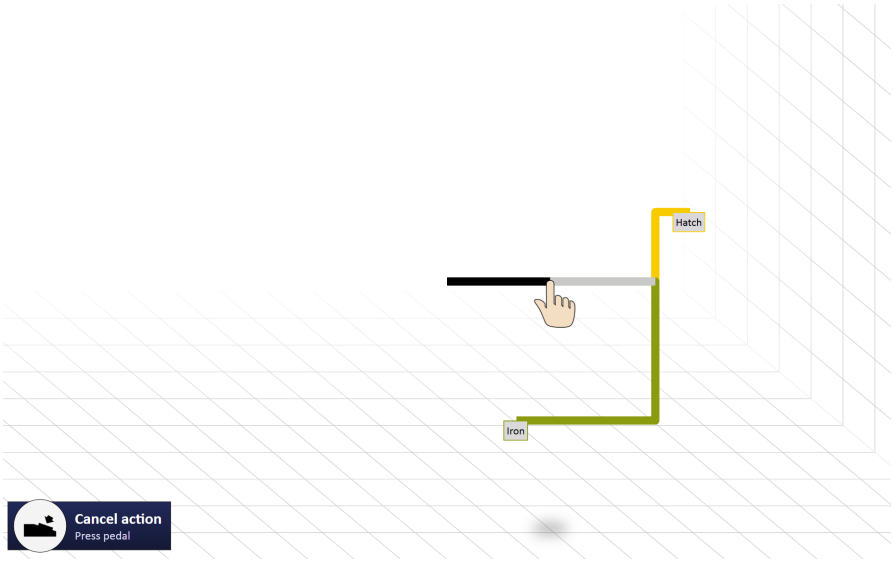
FIGUUR B.3: Gebruiker houdt één hand omhoog

Wanneer gebruikers signaleren dat ze een gebaar willen starten, verschijnt er een virtuele representatie van hun hand op het scherm en kunnen ze een van de getoonde gebaren uitvoeren. In Figuur B.4 wordt de beginstaat van de visualisatie getoond.



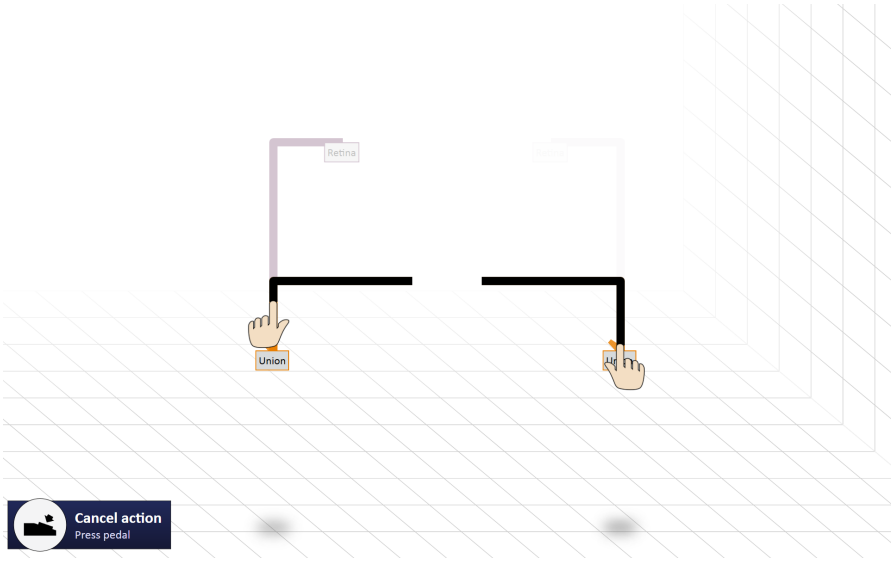
FIGUUR B.4: Gebruiker start uitvoering van unimanueel gebaar

Naarmate de handcursor een van de getoonde pijlen nadert, zal deze pijl opzwellen en zwart worden. Uiteindelijk verdwijnen de pijlen en wordt er overgeschakeld naar een tweede visualisatie. Hierin worden volledige gebaarpaden getoond, maar enkel van de gebaren waar gebruikers vanuit hun cursor nog naartoe kunnen. Figuur B.5 toont een voorbeeldsituatie die aansluit op de vorige afbeelding.



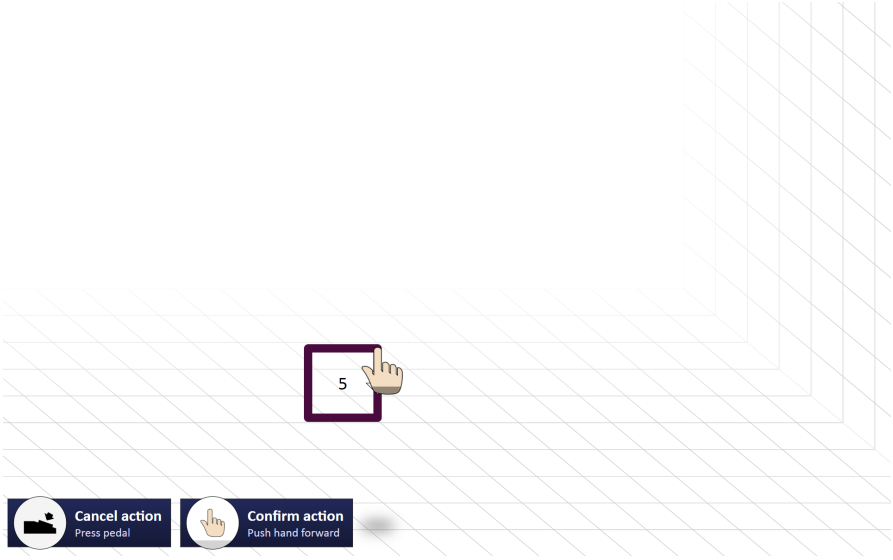
FIGUUR B.5: Gebruiker beweegt hand voorbij de pijl aan de rechterkant

Het systeem kan ook met twee handen tegelijk gebruikt worden. Zulke bewegingen zijn complexer dan eenhandige bewegingen omdat er een niveau van coördinatie moet zijn tussen de bewegingen van beide handen. In Figuur B.6 is een gebruiker een bimanueel gebaar aan het uitvoeren.



FIGUUR B.6: Gebruiker voert een bimanueel gebaar uit

Verder hebben we nog twee speciale gebaartypes voorzien: lussen en sliders. Figuur B.7 en B.8 tonen twee voorbeelden: een unimanuele lus en een bimanuele slider.



FIGUUR B.7: Gebruiker voert een unimanueel lusgebaar uit

FIGUUR B.8: Gebruiker voert een bimanueel slidergebaar uit

Tijdens gebruikerstesten vroegen we 15 deelnemers om een set van 18 gebaren uit te voeren. 14 deelnemers hebben met succes de volledige takenset kunnen uitvoeren door gebruik te maken van onze visualisatie. We konden uit de test opmaken dat ons systeem een veelbelovende eerste stap zet naar een oplossing voor het discoverabilityprobleem dat typerend is aan mid-air gestures, maar o.a. op vlak van dieptecues zal er nog meer onderzoek nodig zijn om systeemfeedback te verbeteren.

Bibliography

[1] Fraser Anderson and Walter Bischof. "Learning and Performance with Gesture Guides". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013, pp. 1109–1118.

[2] *Android 2.2 Platform Highlights*. URL: https://developer.android.com/about/versions/android-2.2-highlights.html (visited on 10/28/2013).

[3] *AutoCAD*. URL: http://www.autodesk.com/products/autodesk-autocad/overview (visited on 10/22/2013).

[4] Olivier Bau, Emilien Ghomi, and Wendy Mackay. *Arpege: Design and Learning of multi-finger chord gestures*. Tech. rep. University of Paris-Sud, 2010.

[5] Olivier Bau and Wendy Mackay. "OctoPocus: A Dynamic Guide for Learning Gesture-Based Command Sets". In: *Proceedings of the 21st annual ACM symposium on User interface software and technology*. 2008, pp. 37–46.

[6] *Big Buck Bunny*. URL: http://www.bigbuckbunny.org/ (visited on 04/03/2014).

[7] Niall Brady. *How can I disable the "While we are getting things ready" animation for All users in Windows 8*. 2012. URL: http://www.niallbrady.com/2012/09/24/how-can-i-disable-the-while-we-are-getting-things-ready-animation-for-all-users-in-windows-8/ (visited on 10/28/2013).

[8] Andrew Bragdon, Arman Uguray, Daniel Wigdor, Stylianos Anagnostopoulos, Robert Zeleznik, and Rutledge Feman. "Gesture Play: Motivating Online Gesture Learning with Fun, Positive Reinforcement and Physical Metaphors". In: *ACM International Conference on Interactive Tabletops and Surfaces*. ACM, 2010, pp. 39–48.

[9] Andrew Bragdon, Robert Zeleznik, Brian Williamson, Timothy Miller, and Joseph LaViola Jr. "GestureBar: Improving the Approachability of Gesture-based Interfaces". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 2269–2278.

[10] Peter Brandl, Clifton Forlines, Daniel Wigdor, Michael Haller, and Chia Shen. "Combining and Measuring the Benefits of Bimanual Pen and Direct-Touch Interaction on Horizontal Interfaces". In: *Proceedings of the working conference on Advanced visual interfaces*. ACM, 2008, pp. 154–161.

[11] John M. Carroll and Mary Beth Rosson. In: *Interfacing thought: cognitive aspects of human-computer interaction*. Ed. by John M. Carroll. 1987. Chap. Paradox of the active user, pp. 80–111.

[12] *CC0 1.0 Universal*. URL: http://creativecommons.org/publicdomain/zero/1.0/ (visited on 04/03/2014).

[13] Victor Cheung, Jens Heydekorn, Stacey Scott, and Raimund Dachselt. "Revisiting Hovering: Interaction Guides for Interactive Surfaces". In: *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces*. ITS '12. Cambridge, Massachusetts, USA: ACM, 2012, pp. 355–358. ISBN: 978-1-4503-1209-7. DOI: 10.1145/2396636.2396699. URL: http://doi.acm.org/10.1145/2396636.2396699.

[14] *Chrome Help*. URL: https://support.google.com/chrome/?hl=en (visited on 09/26/2013).

[15] *Civilization V*. 2010. URL: http://www.civilization5.com/ (visited on 10/03/2013).

[16] *Company of Heroes*. URL: http://www.companyofheroes.com/ (visited on 11/23/2013).

[17] *Designing Context-Sensitive Help*. 2012. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ms670137%28v=vs.85%29.aspx (visited on 09/26/2013).

[18] Paul Dietz and Darren Leigh. "DiamondTouch: a multi-user touch technology". In: *Proceedings of the 14th annual ACM symposium on User interface software and technology*. ACM. 2001, pp. 219–226.

[19] Tom Djajadiningrat, Kees Overbeeke, and Stephan Wensveen. "But how, Donald, tell us how?: on the creation of meaning in interaction design through feedforward and inherent feedback". In: *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*. ACM, 2002, pp. 285–291.

[20] *Download Marble Blast Gold*. URL: http://www.garagegames.com/products/download/15 (visited on 10/03/2013).

[21] David Einstein. "Microsoft's Bob: Quantum leap in computers?" In: *Herald-Journal* (Jan. 1995). Newspaper citation in absence of a primary internet source from Microsoft, B8.

[22] Jennifer Fernquist, Tovi Grossman, and George Fitzmaurice. "Sketch-Sketch Revolution: An Engaging Tutorial System for Guided Sketching and Application Learning". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 373–382.

[23] *File:Clippy-letter.PNG*. 2005. URL: https://en.wikipedia.org/wiki/File:Clippy-letter.PNG (visited on 09/27/2013).

[24] *Flipboard*. URL: https://flipboard.com/ (visited on 10/03/2013).

[25] Dustin Freeman, Hrvoje Benko, Meredith Morris, and Daniel Wigdor. "ShadowGuides: visualizations for in-situ learning of multi-touch and whole-hand gestures". In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*. ACM, 2009, pp. 165–172.

[26] Dustin Freeman, Hrvoje Benko, Meredith Morris, and Daniel Wigdor. *ShadowGuides: Visualizations for In-Situ Learning of Multi-Touch and Whole-Hand Gestures*. 2009. URL: https://www.youtube.com/watch?v=0faNHo5q38s (visited on 10/21/2013).

[27] Emilien Ghomi, Stéphane Huot, Olivier Bau, Michel Beaudouin-Lafon, and Wendy Mackay. "Arpège: learning multitouch chord gestures vocabularies". In: *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces*. ACM, 2013, pp. 209–218.

[28] Trevor Grayling. "If We Build It, Will They Come? A Usability Test of Two Browser-based Embedded Help Systems". In: *Technical Communication* 49.2 (May 2002), pp. 193–209.

[29] Tovi Grossman and George Fitzmaurice. "ToolClips: An Investigation of Contextual Video Assistance for Functionality Understanding". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 1515–1524.

[30] *GuildWars2.com*. URL: https://www.guildwars2.com/ (visited on 10/03/2013).

[31] Juan David Hincapié-Ramos, Xiang Guo, Paymahn Moghadasian, and Pourang Irani. "Consumed Endurance: A metric to quantify arm fatigue of mid-air interactions". In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM. 2014, pp. 1063–1072.

[32] Eric Horvitz. "Lumiere Project: Bayesian Reasoning for Automated Assistance". In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. 1998, pp. 256–265.

[33] *How To Implement "What's This?" Help in Visual Basic*. 2004. URL: http://support.microsoft.com/kb/142249 (visited on 09/26/2013).

[34] *IKVM.NET Home Page*. URL: http://www.ikvm.net/ (visited on 03/14/2014).

[35] *Implementing "What's This?" WinHelp*. URL: http://msdn.microsoft.com/en-us/library/aa975791%28v=vs.71%29.aspx (visited on 09/26/2013).

[36] *IntelliJ IDEA – The Best Java and Polyglot IDE*. URL: http://www.jetbrains.com/idea/ (visited on 10/09/2013).

[37] *Intuit Quickbooks*. URL: http://quickbooks.intuit.com/ (visited on 11/07/2013).

[38] *iOS 7 - Siri*. URL: https://www.apple.com/ios/siri/ (visited on 11/17/2013).

[39] Shahram Izadi, Harry Brignull, Tom Rodden, Yvonne Rogers, and Mia Underwood. "Dynamo: A Public Interactive Surface Supporting the Cooperative Sharing and Exchange of Media". In: *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*. UIST '03. Vancouver, Canada: ACM, 2003, pp. 159–168. ISBN: 1-58113-636-6. DOI: 10.1145/964696.964714. URL: http://doi.acm.org/10.1145/964696.964714.

[40] *Kinect*. URL: http://www.xbox.com/en-US/kinect (visited on 02/27/2014).

[41] *Kinect for Windows Dev Center*. URL: http://www.microsoft.com/en-us/kinectforwindowsdev/default.aspx (visited on 03/14/2014).

[42] *Kinect Sensor*. URL: http://msdn.microsoft.com/en-us/library/hh438998.aspx (visited on 06/03/2014).

[43] *KinectInteraction Concepts*. URL: http://msdn.microsoft.com/en-us/library/dn188673.aspx (visited on 04/28/2014).

[44] P. O. Kristensson and L. C. Denby. "Continuous Recognition and Visualization of Pen Strokes and Touch-screen Gestures". In: *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*. SBIM '11. Vancouver, British Columbia, Canada: ACM, 2011, pp. 95–102. ISBN: 978-1-4503-0906-6. DOI: 10.1145/2021164.2021181. URL: http://doi.acm.org/10.1145/2021164.2021181.

[45] Myron W Krueger, Thomas Gionfriddo, and Katrin Hinrichsen. "VIDEOPLACE—an artificial reality". In: *ACM SIGCHI Bulletin*. Vol. 16. 4. ACM. 1985, pp. 35–40.

[46] Gordon Kurtenbach and Wiliam Buxton. "User learning and performance with marking menus". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1994, pp. 258–264.

[47] Gordon Kurtenbach, Abigail Sellen, and William Buxton. "An empirical evaluation of some articulatory and cognitive aspects of marking menus". In: *Human-Computer Interaction* 8 (1 Mar. 1993), pp. 1–23.

[48] Wei Li, Tovi Grossman, and George Fitzmaurice. "GamiCAD: A Gamified Tutorial System For First Time AutoCAD Users". In: *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 2012, pp. 103–112.

[49] Hsiang-Sheng Liang, Kuan-Hung Kuo, Po-Wei Lee, Yu-Chien Chan, Yu-Chin Lin, and Mike Chen. "SeeSS: Seeing What I Broke – Visualizing Change Impact of Cascading Style Sheets (CSS)". In: *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 2013, pp. 353–356.

[50] Allan Christian Long Jr. "Quill: a gesture design tool for pen-based user interfaces". PhD thesis. University of California, 2001.

[51] Erich Luening. *Microsoft tool "Clippy" gets pink slip*. 2001. URL: http://news.cnet.com/2100-1001-255671.html (visited on 09/27/2013).

[52] I.S. MacKenzie. *Human-Computer Interaction: An Empirical Research Perspective*. Elsevier Science, 2012. ISBN: 9780124071650. URL: https://encrypted.google.com/books?id=k0kBgyCaokAC.

[53] Justin Matejka, Tovi Grossman, and George Fitzmaurice. "Ambient Help". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 2751–2760.

[54] *Microsoft HTML Help 1.4*. 2012. URL: http://msdn.microsoft.com/en-us/library/ms670169%28v=vs.85%29.aspx (visited on 09/26/2013).

[55] *Microsoft Office 97 Released to Manufacturing*. 1996. URL: https://www.microsoft.com/en-us/news/press/1996/nov96/rtmpr.aspx (visited on 09/27/2013).

[56] *Microsoft Word 2000: Using What's This? and ScreenTips*. URL: http://learningfast.com.au/cbts/word2000/activities/html/114.htm (visited on 09/26/2013).

[57] Meredith Ringel Morris, Anqi Huang, Andreas Paepcke, and Terry Winograd. "Cooperative Gestures: Multi-user Gestural Interactions for Co-located Groupware". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '06. Montréal, Québec, Canada: ACM, 2006, pp. 1201–1210. ISBN: 1-59593-372-7. DOI: 10.1145/1124772.1124952. URL: http://doi.acm.org/10.1145/1124772.1124952.

[58] Mathieu Nancel, Julie Wagner, Emmanuel Pietriga, Olivier Chapuis, and Wendy Mackay. "Mid-air pan-and-zoom on wall-sized displays". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2011, pp. 177–186.

[59] *New Super Mario Bros... : Walkthrough - Part 1 w/ BrendanGaming*. 2011. URL: https://www.youtube.com/watch?v=zS3gTU8Ox4k (visited on 10/11/2013).

[60] Jacob Nielsen. *10 Usability Heuristics for User Interface Design*. 1995. URL: http://www.nngroup.com/articles/ten-usability-heuristics/ (visited on 10/03/2013).

[61] Jakob Nielsen. *Usability 101: Introduction to Usability*. 2012. URL: http://www.nngroup.com/articles/usability-101-introduction-to-usability/ (visited on 10/09/2013).

[62] Donald Norman. *The Psychology of Everyday Things*. Basic Books, 1988.

[63] Donald A Norman and Jakob Nielsen. "Gestural interfaces: a step backward in usability". In: *interactions* 17.5 (2010), pp. 46–49.

[64] *Office.com*. URL: http://office.microsoft.com/en-us/ (visited on 10/07/2013).

[65] *Official Site - New Super Mario Bros. Wii*. 2009. URL: http://www.mariobroswii.com/ (visited on 10/11/2013).

[66] *"ok glass, "* URL: https://support.google.com/glass/answer/3079305 (visited on 11/17/2013).

[67] *Paint.NET*. URL: http://www.getpaint.net/ (visited on 10/03/2013).

[68] *Plugins/Thumbnail*. 2008. URL: http://wiki.compiz.org/Plugins/Thumbnail (visited on 10/07/2013).

[69] *Portal*. URL: http://www.valvesoftware.com/games/portal.html (visited on 11/17/2013).

[70] Chris Pratley. *Clippy and User Experiences*. 2004. URL: http://blogs.msdn.com/b/chris_pratley/archive/2004/05/05/126888.aspx (visited on 09/27/2013).

[71] *QuickBooks®in the Classroom: Student's Resource Guide*. Intuit Canada. 2005.

[72] Raf Ramakers, Davy Vanacken, Kris Luyten, Karin Coninx, and Johannes Schöning. "Carpus: a non-intrusive user identification technique for interactive surfaces". In: *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM. 2012, pp. 35–44.

[73] Stephanie Rosenbaum, Laurie Kantner, and Garett Dworman. "Helping Users to Use Help: Results from Two International Conference Workshops". In: *International Professional Communication Conference*. 2005, pp. 181–187.

[74] Alexander Serenko, Nick Bontis, and Brian Detlor. "End-user adoption of animated interface agents in everyday work applications". In: *Behaviour and Information Technology* (2007), pp. 119–132.

[75] Rajinder Sodhi, Hrvoje Benko, and Andrew Wilson. "LightGuide: Projected Visualizations for Hand Movement Guidance". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. Austin, Texas, USA: ACM, 2012, pp. 179–188. ISBN: 978-1-4503-1015-4. DOI: 10.1145/2207676.2207702. URL: http://doi.acm.org/10.1145/2207676.2207702.

[76] *Spy Fox Dry Cereal - PC/Mac*. Amazon link in absence of an official website. 2002. URL: http://www.amazon.com/Spy-Fox-Dry-Cereal-PC-Mac/dp/B00006BN8X (visited on 10/03/2013).

[77] *SuSE Linux Administration Guide*. For SuSE Linux 9.2. Novell Inc. 2004.

[78] *TalkBack*. URL: https://play.google.com/store/apps/details?id=com.google.android.marvin.talkback (visited on 10/08/2013).

[79] *Tracking Users with Kinect Skeletal Tracking*. URL: http://msdn.microsoft.com/en-us/library/jj131025.aspx (visited on 02/27/2014).

[80] *Tracking Users with Kinect Skeletal Tracking*. URL: http://msdn.microsoft.com/en-us/library/jj131025.aspx (visited on 03/14/2014).

[81] Martijn Van Welie. *Interaction Design Pattern Library*. URL: http://www.welie.com/patterns/index.php (visited on 11/19/2013).

[82] Davy Vanacken. "Touch-based Interaction and Collaboration in Walk-up-and-use and Multi-user Environments". PhD thesis. Hasselt University, 2012.

[83] Davy Vanacken, Kris Luyten, and Karin Coninx. *TouchGhosts: Guides for Improving Visibility of Multi-Touch Interaction*. Tech. rep. Hasselt University, 2009.

[84] Daniel Vogel and Ravin Balakrishnan. "Interactive public ambient displays: transitioning from implicit to explicit, public to personal, interaction with multiple users". In: *Proceedings of the 17th annual ACM symposium on User interface software and technology*. ACM, 2004, pp. 137–146.

[85] *VoiceOver for iOS*. URL: https://www.apple.com/accessibility/ios/voiceover/ (visited on 10/08/2013).

[86] Robert Walter, Gilles Bailly, and Jörg Müller. "StrikeAPose: Revealing Mid-air Gestures on Public Displays". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '13. Paris, France: ACM, 2013, pp. 841–850. ISBN: 978-1-4503-1899-0. DOI: 10.1145/2470654.2470774. URL: http://doi.acm.org/10.1145/2470654.2470774.

[87] *Welcome to Microsoft(R) Agent*. 2009. URL: https://www.microsoft.com/products/msagent/ (visited on 09/27/2013).

[88] D. Wigdor and D. Wixon. *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. Elsevier Science, 2011. ISBN: 9780123822321. URL: https://encrypted.google.com/books?id=ID0L0EI79-YC.

[89] *WinZip for Windows, Mac and Mobile*. URL: http://www.winzip.com/ (visited on 10/03/2013).

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Designing Intelligible Visualizations To Accommodate Mid-Air Gesture Input

Richting: **master in de informatica-Human-Computer Interaction**
Jaar: **2014**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

De Decker, Pieter

Datum: **10/06/2014**