



Microsoft Surface™ User Experience Guidelines

Designing for Successful Touch Experiences

Version 1.0
June 30, 2008

Copyright

This document is provided for informational purposes only, and Microsoft makes no warranties, either express or implied, in this document. Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results from the use of this document remains with the user. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, financial and other data, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, places, financial or other data, or events is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Excel, Microsoft Surface, the Microsoft Surface logo, Segoe, Silverlight, Windows, Windows Media, Windows Vista, Virtual Earth, Xbox LIVE, XNA, and Zune are either registered trademarks or trademarks of the Microsoft group of companies.

Apple and iPod are trademarks of Apple Inc., registered in the U.S. and other countries.

Bluetooth® is a registered trademark of Bluetooth SIG, Inc., and is used under license from Bluetooth.

All other trademarks are property of their respective owners.

Contents

1.0	Introduction	6
1.1	Who this document is intended for	6
1.2	How this document is organized	6
2.0	Vision and Philosophy	7
2.1	Vision	7
2.2	Philosophy	7
2.2.1	From Command Line Interfaces to Natural User Interfaces	8
2.2.2	Designing Immersive Experiences	9
2.2.3	Benefits of Touch and Gesture for Microsoft Surface Experiences	10
2.2.4	Designing for Object Interaction	10
3.0	Principles and Benefits	11
3.1	Design Principles	11
3.1.1	Seamless	11
3.1.2	Social	13
3.1.3	Spatial	15
3.1.4	Superrealism	18
3.1.5	Contextual Environments	19
3.1.6	Scaffolding	21
3.1.7	Performance Aesthetics	22
3.1.8	Direct Manipulation	24
3.2	Key User Experience Benefits	25
3.2.1	Microsoft Surface Is Natural and More Intuitive	26
3.2.2	Microsoft Surface Is Unique and Magical	26
3.2.3	Microsoft Surface Is Premium and Authentic	26
3.2.4	Microsoft Surface Is Aware and Responsive	27
3.2.5	Microsoft Surface Is Social and Collaborative	27
4.0	User Interface Overview	28
4.1	Putting the Philosophy and Principles into Practice	28
4.2	Considerations	28
4.2.1	Project Considerations	28
4.2.2	What Qualifies as a Good Microsoft Surface Application?	30
4.2.3	Social	32
4.2.4	Object and Device Interaction	34
4.2.5	End User Considerations	39
4.2.6	Business Considerations	40
4.2.7	Brand Considerations	40
4.2.8	Microsoft Surface Software Environment	41
5.0	Interaction Design	56
5.1	Approach	56
5.2	Guidelines	57
5.2.1	Seamless	57
5.2.2	Social	58
5.2.3	Spatial	59
5.2.4	Super-Realism	60

5.2.5	Contextual Environments	61
5.2.6	Scaffolding	62
5.2.7	Performance Aesthetics	63
5.2.8	Direct Manipulation	64
5.3	Gestures	65
5.3.1	Gesture Parameters	65
5.3.2	Enabled Gestures	66
6.0	Visual Design	74
6.1	Approach	74
6.1.1	Creating Visual Design Vocabularies	74
6.1.2	Visual Design Attributes	75
6.2	Guidelines	77
6.2.1	Visual Branding Principles	77
6.2.2	Layout	78
6.2.3	Depth	82
6.2.4	Shape and Form	87
6.2.5	Texture	89
6.2.6	Colors	92
6.2.7	Typography	93
6.2.8	Iconography	94
6.2.9	Motion Design	97
7.0	Physical Design	102
7.1	Industrial Design	102
7.1.1	Approach	102
7.1.2	Guidelines	104
7.2	Environment Design	106
7.2.1	Approach	106
7.2.2	Guidelines	106
8.0	The Microsoft Surface User Interface	110
8.1	User Interface Components	110
8.1.1	Attract Application	110
8.1.2	Launcher	116
8.1.3	Notifications	119
8.1.4	Out of Order Screen	123
8.1.5	Application Backgrounds	123
8.1.6	User Assistance / Help	124
8.2	User Interface Language	126
8.3	User Interface Controls	137
8.3.1	Alternative Designs	138
8.3.2	Redesigning Existing Controls	138
8.3.3	Creating All-New Controls	139
8.3.4	ScatterView	140
8.3.5	SurfaceScrollViewer	144
8.3.6	TagVisualizer	147
8.3.7	SurfaceInkCanvas	149
8.3.8	Stack & Scroller	150
8.3.9	SurfaceButton	154
8.3.10	SurfaceCheckBox	160

8.3.11	SurfaceRadioButton	165
8.3.12	SurfaceSlider	170
8.3.13	SurfaceScrollBar	175
8.3.14	SurfaceListBox	179
8.3.15	SurfaceTextBox	183
8.3.16	SurfacePasswordBox	191
9.0	Methods and Tools	192
9.1	Research	192
9.1.1	Evaluating a Microsoft Surface Experience	192
9.1.2	Exploratory Research	192
9.1.3	Generative Research	193
9.1.4	Evaluative Research	193
9.1.5	Implications to the Research Flow	194
9.1.6	Research Methods	196
9.1.7	Research Tools and Techniques	199
9.2	Design	203
9.2.1	Design Methods	203
9.2.2	Implications to the Design Flow	209
10.0	References	211

1.0 Introduction

1.1 Who this document is intended for

This document is intended for designers, program managers, and developers of Touch- and Gesture-enabled applications, in particular for the Surface Computing initiative at Microsoft®. The document is written to advise those individuals about the unique opportunities afforded by the technology, and to convey some of the learnings of the Microsoft Surface™ (as used in this document, “Microsoft Surface” or “Surface”) team gleaned during its development over the last several years. This document is not a recipe book or prescription that outlines exact specifications of Surface applications. Instead, it aims to inspire an approach to designing a great Surface experience by informing you about the philosophy of the Surface product and providing practical design information on key building blocks.

Designing Surface experiences that incorporate compelling touch solutions requires a different approach to thinking about user interface (UI) from the development of traditional graphical user interface (GUI) products or Web experiences.

1.2 How this document is organized

This document is organized into the following sections:

Section 2: Vision and Philosophy describes our surface computing vision and our Natural User Interface philosophy.

Section 3: Principles and Benefits describes our core design principles behind great Surface experiences, and the key user benefits that result from surface computing in general.

Section 4: User Interface Overview looks at what is involved in creating Surface experiences, providing a broad overview of how to manage Surface project needs, end user goals, branding requirements, and business objectives.

Section 5: Interaction Design, Section 6: Visual Design, and Section 7: Physical Design break down the Surface philosophy and principles into actionable advice for designers. Each section describes how to approach problem solving for Surface experiences, and offers guidelines and examples for each creative discipline.

Section 8: The Microsoft Surface User Interface lists the common user interface components and controls needed to build Surface experiences, as well as a treatise on how to sculpt and deploy copy to best suit the “voice” of Surface.

Section 9: Methods and Tools describes the ways that research supports an iterative design flow for Surface experiences. It is organized into areas including research and design methods, and tools and techniques.

Section 10: References offers helpful references for readers who want to delve deeper into Surface-related concepts and topics.

2.0 Vision and Philosophy



"Any sufficiently advanced technology is indistinguishable from magic." -Arthur C. Clarke

2.1 Vision

Imagine using technology without ever seeing the technology. No tech-talk, tech-tasks, tech-knowledge, tech-management, tech-worries, and so on. Instead, the "tech-tosterone" is truly invisible, hidden below the surface and not taking center-stage. Above the surface, you engage with quality information, style, and a killer experience. This is the promise of Microsoft Surface.

This experience is not about what technology can do for you, but instead how technology can conform to your needs. This experience is about how technology can remove constraints and create freedom that enables you to engage in more natural, fun, and amazing ways. That freedom changes everything.

The "wow factor" of a computing experience is transformed into a "big-bang factor." It is so powerful that your first surface computing experience will spark an explosion of possibilities in your mind. Content is manipulated directly, naturally, and without fear of error. This instant sense of control releases your imagination and creates a desire to do everything with such ease and delight. You are actually giddy with a sense of wonder and magic.

This sense of wonder comes from being freed from complexity, and being released from constantly keeping pace with the rapid technological advances that widen the gap between a device and you. With Surface, these technological advances are used in ways to simplify the human-computer interaction and close this gap. All people, regardless of technical knowledge, should be able to easily enjoy and benefit from a Surface experience. With Surface, you are always in control and technology is racing to keep pace with your imagination.

These new digital interactions are as rich, detailed, and responsive as the real-world...and even more so. Boundaries between physical and digital are blurred, and real-world details have a humble but effective presence within digital interactions. Imagine having all of this but also being unconstrained by it. That additional freedom is about real-world experiences with digital benefits.

When we are successful in achieving all of this, users will apply their life-knowledge instead of tech-knowledge to easily enjoy our solutions. The user interface will vanish and it will become instead about the users, their goals, and their experiences.

Every user experience (UX) principle and guideline in this document is working toward creating such great experiences.

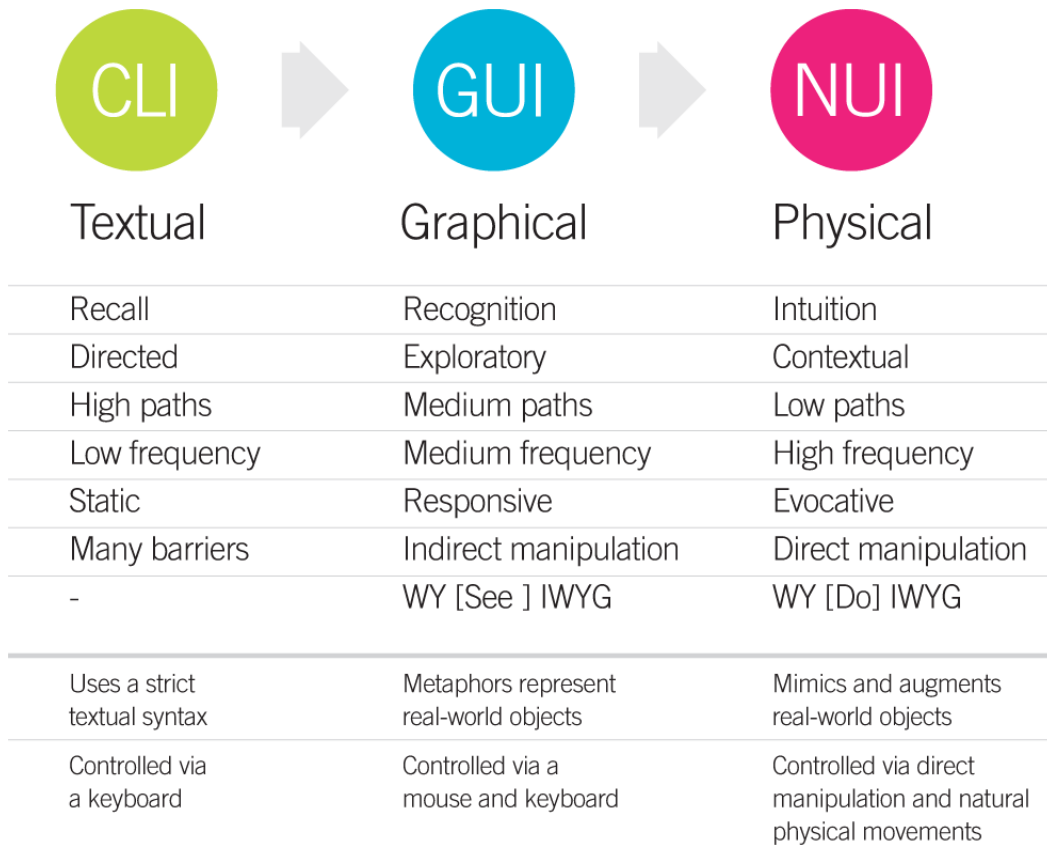
2.2 Philosophy

Designing for touch and gesture is not about doing old things in new ways; it is about doing new things naturally. Touch and gesture experiences are not about retrofitting existing experiences to enable new interaction models. It is about inventing and implementing new solutions that go beyond existing metaphors and conventions, and leveraging the full spectrum of what is possible with the new interaction model and the technology.

In short, Surface computing is about completely redefining the experiences of existing scenarios through touch and gesture.

2.2.1 From Command Line Interfaces to Natural User Interfaces

The start of almost every successful technological innovation is marked by an inflection point, which defines a moment of dramatic change. In the evolution of Human Computer Interaction (HCI), that moment has arrived as experiences evolve from Command Line Interfaces (CLI) to Graphical User Interfaces (GUI) to Natural User Interfaces (NUI).



The advent of Command Line Interfaces (CLI) marked the first inflection point in the history of HCI. CLI provided the most efficient means of commanding and controlling computers by typing text. This form of interaction required people to follow a strict, text-based syntax, memorize the commands they needed to use, and input those commands in a linear sequence. The method of input was the computer keyboard, which was already familiar because it hearkened back to typewriter keyboards and the QWERTY layout introduced in the 1870s.

But people needed a more efficient way of interacting with large amounts of text and numerical data.

Nearly a century later, the mouse was invented in the 1960s to control the motion of a pointer on a screen and an entirely new vernacular was introduced to HCI: Click, drag, double-click, scroll, chord, highlight, roll over, drop, etc. Computer graphics replaced lines of text and metaphors represented real-world objects: Desktops, trash cans, folders, files, etc. People no longer had to memorize commands because they could see the objects they wanted to interact with on the screen. Instead, they began to explore these objects

and recognize the commands they needed to complete their tasks. The experience became exploratory and responsive to what they saw.

This combination of the mouse and computer graphics marked another inflection point in HCI: the Graphical User Interface (GUI). A set of standard controls was developed for this type of interface: Windows, Icons, Menus, and Pointers (WIMP). From the perspective of designing GUI experiences, GUI and WIMP went hand in hand.

Today, with the introduction of touch and gesture, the way that people interact with computers is moving away from manufactured devices, such as mice and keyboards, and toward the natural use of fingers and hands. This newest inflection point in HCI, where interfaces are controlled via natural physical behaviors and movements, is called the Natural User Interface (NUI). The point-and-click vocabulary of GUI is replaced with words that describe real-world gestures: tap, flick, squeeze, pinch, push, scoop, slide, etc. With NUIs, human intuition replaces the need to recall commands in CLI and recognize visual metaphors in GUI. The interface is contextual to the intention and behavior of the person using it. The “what you see is what you get” (WYSIWYG) nature of GUI is replaced with “what you *do* is what you get” (WYDIWYG) in NUI.

While NUI introduces a new way of interacting with computers, it is not the best solution for every task. Each type of interface is more appropriate than the others for a specific context. That is, the best solution depends on what people want to do, why they want to do it, and where they are at the time in the experience. CLI was the most efficient means of interacting with individual commands; GUI made interacting with large amounts of content visual and easy; and now NUI enhances the experience through touch, gesture, and direct manipulation.

2.2.2 Designing Immersive Experiences

There are few examples of designing immersive software experiences as successful as those in the computer and video game industry. The discipline of game design and development has matured as the industry and the technology matured. Many early computer adventure games were fairly linear in design, featured text or static graphics, and were limited to one player, echoing the CLI interaction paradigm of the time.

As graphics technology enabled fast, new 2-D graphics and animation, many games echoed a more exploratory approach to design. They began to utilize the mouse or joystick for indirect manipulation of characters and even mimicked many GUI design patterns with the addition of graphical on-screen menus and two-dimensional radar maps. The focus of these games was the player against the system in a predominantly single-player or solitary experience.

As with Natural User Interfaces (NUI), the majority of the latest generation console video games, such as the Nintendo® Wii™, are fully three-dimensional and involve both spatial and highly intuitive inputs. These types of gestural remotes feature sensory force feedback, incredible interactive responsiveness, immersive sound effects, and super-realistic graphics. The production values of many modern video games parallel those of the motion picture industry with an incredible focus on making the experiences as polished and immersive as possible. A significant investment is expended in making game environments evocative to encourage exploration, as well as making game challenges more open-ended. The shift from single-player games to social games (played cooperatively and competitively at the same time) in a multiplayer community has had a dramatic impact on the ways that designers and developers approach the challenge of designing delightful, engaging, and complex interactive experiences.

While games are not the explicit focus of this document, the process of building playful, rich, interactive

experiences that engage people is similar to the process of designing Surface experiences. Thus important lessons from game design are included in the Philosophy and Principles sections of this document.

2.2.3 Benefits of Touch and Gesture for Microsoft Surface Experiences

The many benefits of designing experiences that are optimized for touch, gesture, and direct manipulation are illustrated throughout this document. Following are just a few of the important benefits to know.

Intuitive expertise

The nature of direct manipulation requires objects and content to behave the way they do in the real world. It results in highly intuitive and error-free interactions that seem like second nature to people, making them experts from the first touch. It also reduces the learning curve significantly and helps people feel as if they can do anything comfortably, without making mistakes.

Powerful, emotional brand builder

The intuitive and direct nature of Surface experiences helps people feel mastery and some sense of immediate satisfaction. These feelings quickly establish a relationship with the customer, one that is engaging and pleasurable. When combined with great design the result can be a strong emotional connection to the brand being represented in the experience. For some time to come, the novelty of Surface experiences will generate significant excitement and desire to “touch” and play with the Surface unit.

Emotional bond

Surface experiences create a strong emotional bond to the people that interact with them. The effect is magical and addictive, which enhances the brand experience and broadens the audience.

Responsiveness

Good Surface experiences are extremely responsive to user interaction, allowing subtle and minute gestures to have an immediate effect on the application. The feedback loop is so tight (enhanced because of the directness of touch) that simply interacting with things feels magical.

2.2.4 Designing for Object Interaction

Surface experiences can recognize many additional devices such as cameras or cell phones, both optically and through Bluetooth® and other networking technologies. This allows for Surface experiences to be part of a much broader set of experiences than a traditional touch kiosk. With the thoughtful interaction between devices, gestures, and large visual displays, amazing retail experiences are immediately possible.

3.0 Principles and Benefits



3.1 Design Principles

There are several key principles at the core of the Microsoft Surface philosophy, which all successfully designed Surface experiences embody. They are timeless, and apply to Surface experiences regardless of release. These principles are intended to guide the design of touch and gesture experiences. More importantly, they encourage you to exploit the possibilities of Surface computing in a way that makes GUI a thing of the past and empowers you to create natural user interfaces that invite people to work or play together through direct manipulation.

Remember that it is not about doing old things in new ways; it is about doing new things naturally.

The following design principles will help you design successful Surface experiences:

- Seamless
- Social
- Spatial
- Superrealism
- Contextual Environments
- Scaffolding
- Performance Aesthetic
- Direct Manipulation

3.1.1 Seamless

The Suspension of Disbelief

Seamless experiences require the user to be mentally and emotionally immersed in order to fearlessly commit to new experiences. This can be achieved by suspending the user's sense of disbelief.

The suspension of disbelief refers to the willingness of a person to accept something as true even if it is fantastic or impossible in the real world. The act of suspending one's disbelief creates an environment that the brain will immediately accept as sufficiently real. It is one of the many appeals of movies, books, plays and other traditional forms of entertainment. Interactivity makes this environment seem closer to real life, and it is the reason why the game industry has focused for many years on building immersive worlds replete with dynamic simulations of a living, breathing environment. Interactivity has gone beyond just simulating the world to creating a better, more fun-filled world.

Microsoft Surface experiences invite the suspension of disbelief by blurring the line between the real and the virtual, combining the effortlessness of manipulating real-world objects with the power of virtual-world capabilities. Imagine a map on a Surface screen. Initially, it may appear as a globe that you spin by flicking it with your finger. You might touch a location to transform the globe into a map of a country, and touch the map to zoom in on a city you want to visit. Each touch zooms in further on the map until you see points of interest that you touch to create a personalized itinerary. Obviously, the Surface unit is not a

globe capable of spinning or a map capable of creating itineraries. But disbelief is suspended when the rendition of the visible object mimics its real-world counterpart to such an extent that people believe it to be real as well.

Microsoft Surface experiences not only mimic real-world objects, but also leverage the computing capabilities to extend them beyond what is possible in the real world. To achieve this marriage of the physical and virtual worlds successfully, the line between them must be seamless and the performance of the technology must be flawless.

Fingers and objects

Microsoft Surface experiences offer direct manipulation of fingers and physical objects. The physical objects can interact with the Surface experience, making the experience feel seamless between the physical and virtual. This seamlessness can be achieved with tagged, Bluetooth®- or Wi-Fi®-enabled devices.

Untagged

Microsoft Surface experiences sense the presence of any object that is placed on the screen. They respond to an untagged object similarly to the way they respond to an entire hand placed on the screen. When an object touches the Surface screen, the response is immediate and the line between the physical object and the virtual experience is blurred. Because Surface cannot identify a specific untagged object, the feedback is whimsical and magical as opposed to informational. For example, Surface might respond to an untagged glass placed on the screen by animating the background or rings around the glass.

But because Surface can recognize tagged Bluetooth®- and Wi-Fi enabled devices, its response can be more meaningful and useful.

Tagged

A Byte Tag is a barcode-like label that is adhered to an object (see [User Interface Overview, Object and Device Interaction](#)). When the “tagged” object is placed on the Surface screen, Surface “reads” the Byte Tag and responds to the object depending on which type of object it is. For example, a mobile phone retailer might place Byte Tags on cell phones for an in-store retail experience, so that when one of these phones is placed on Surface, the screen displays information about that particular cell phone model.

Bluetooth and Wi-Fi

Devices that are Bluetooth®- and Wi-Fi enabled allow content and data to be pushed to and pulled from Surface experiences. In this way, users can manage content as physical units on Surface, interacting with content directly (such as dragging a virtual file from one physical device to another). Virtual content in Surface experiences can be uploaded to mobile devices, which extends the experience beyond the physical boundaries of the Surface unit itself.

Physical hardware made virtual

The nature of direct manipulation eliminates the need for peripheral input hardware, such as mice, keyboards, and joysticks, because virtual representations of those devices appear directly in the interface. For example, a hardware trackball might be replaced with a virtual trackball that you interact with in the same way. The obvious limitation of virtual hardware is that it does not provide the haptic and physical feedback of its physical counterparts. However, the benefit is added simplicity and purity of experience, with only touch-based gestures to learn, increasing emotional engagement, a sense of wonder, and a natural and direct means of interacting with content.

Physical surroundings

Microsoft Surface experiences can also respond to people and physical objects around the physical unit. For example, if a proximity sensor is added to the Surface unit, the experience can detect whether a person is approaching or leaving the unit (by detecting forward motion vs. backward motion) and respond accordingly. If pressure sensors are added to the seats that surround the unit, the experience can detect the number of people nearby and gather information about them, such as their size, and respond accordingly. Both of these examples help to blur the line between the physical and virtual—indeed, between Surface and its very environment—and prolong the suspension of disbelief.

Audio

Audio can also be used to interact with Surface experiences by instigating immediate feedback or a response. For example, the Surface unit might “listen” for specific spoken words, such as “loud,” in a conversation, and then adjust the sound volume. Voice control is another, less directly physical way to control the experience (based on the tone of the conversation, level of noise, etc.). These examples help create seamless experiences between the audio and the interaction.

Guidelines

The following guidelines govern the design of experiences that adhere to the core idea of seamless.

- Make virtual things behave like real things. A fundamental way to achieve suspension of disbelief is by making objects on the screen behave similarly to the way objects in the real world behave. Subtle physics in motion, inertia of movement, and natural-feeling collisions help to create the sense of illusion when manipulating virtual things in a way that is impossible in reality.
- Erase the line between the physical and virtual. In order to perpetuate the suspension of disbelief, the Surface experience must respond continuously to fingers and physical objects by displaying information on the screen. For example, if a cocktail is placed on a Surface screen, the user interface should respond to it by surrounding the glass with an aura and displaying the name and ingredients of the cocktail (provided a drink-specific Byte Tag is present on the bottom of the glass).
- Make sure that the experience provides interaction feedback on all objects, even if they do not actually do anything. In other words, if an object is not interactive, the experience still provides subtle feedback that acknowledges the object so people are never left wondering whether the object is broken or malfunctioning.

3.1.2 Social

From Solitude to Society

Microsoft Surface experiences remove the physical boundaries between people and encourage as many people as can fit around the Surface unit to interact with it together and at the same time.

In standard GUIs, social barriers are a result of the input and output methods. For example, experiences are inherently single-person when the experience is mediated by only one mouse, one keyboard, and no touch screen. Microsoft Surface, on the other hand, accepts multi-person input. Users can gather around one Surface unit and play the same instance of a game at the same time. Microsoft Surface can elevate activities from a solitary experience to a social experience.

The social experience is not limited to the interactions between people and the Surface interface. The less

communication that happens between an individual and the Surface unit, the more communication happens between the people around it. People focus less on the computer and more on each other, and the computer becomes secondary to the people using it.

In the game world, both competitive and cooperative games have risen to displace the traditional single-player game as the true market exciter. Console games have increasingly become the mediator for social interaction that occurs in the living room between people engaged in the game experience. The point of playing such games and the enjoyment derived from them has shifted away from the game itself and onto the social interaction that they enable. Cooperative techniques from game design can be co-opted directly into the design of Surface experiences to make them more engaging and fun.

Social gestures

Gestures can be categorized into two types:

- Physical gestures are used to interact with the objects and content in Surface experiences. For example, a person might flick a photo to move it from one side of the Surface screen to another.
- Social gestures are used to facilitate social interactions between people. For example, a person might show a photo to friends sitting across the Surface unit by flicking it at them. They attempt to get the other person to do or not do something, such as look, stop, follow, etc.

Another type of gesture—emotional gestures—typically takes place above the Surface unit and express emotions such as joy, anger, etc. In Surface applications that feature life-like entities (e.g., soldiers in a game, social characters representing the brand, etc.), it is important to develop gestures that represent intent. Much exploration remains to be done to assess how a scenario enabled in a Surface experience may reveal the different needs and preferences of people across the world.

While physical gestures may remain fairly consistent across cultures, the properties of objects (e.g., size, shape, contents, and behavior) may vary geographically. For example, the default size of a photo in China is half the size of photos in the U.S.; therefore, photo manipulation may feel different. Hardware appears different in different countries so objects representing all sorts of hardware, from doorknobs to power outlets, may require a different representation, feel, and gesture. These variations present an opportunity to develop and enable a rich set of possible gestures to manipulate objects, especially if those objects have different cultural representations.

Orientation

Surface was intentionally designed to be controlled from many vantage points around the perimeter. Designing a Surface experience that can be freely oriented to any edge (or truly require no specific orientation) should be considered as part of your application's design. Accommodating many people adds significant complexity to the orientation of text and graphics. Ideally, the controls, objects, and labels of an application gracefully and appropriately orient themselves to the positions of the users interacting with them, if possible and where appropriate.

Guidelines

The following guidelines govern the design of experiences that adhere to the core idea of social:

- Prioritize Surface experiences for the social interactions that occur between the people at the Surface unit. This social interaction must be considered for all Surface experiences.
- Be aware of social conversations. There are primary and secondary people in any conversation where the primary person is the one "speaking" and the secondary people are listening. Microsoft

Surface experiences should allow people to engage in “conversations” and quietly step in and out of them. For example, a primary person might be interacting with photos and a secondary person might be observing the interaction.

- Design with the potential for many people interacting with the experience simultaneously. Consider the implications of people touching objects from different sides of the Surface unit and how users from any side might re-orient the touched objects.

3.1.3 Spatial

From Flat to Fat

Traditional GUI experiences are somewhat planar, and objects generally appear flat and two-dimensional. Some objects are made to look as if they have a depth by skewing two-dimensional planes, adding shadows, and overlapping elements. This is known as “two-and-a-half dimensional,” or “2.5D.” Microsoft Surface experiences extrude the visual planes into a true third dimension by leveraging the x, y, and z axes both visually and behaviorally. In other words, objects appear to have volume or take on real-world behaviors three dimensionally and people can navigate spatially in all dimensions. Microsoft Surface experiences go from being flat and linear to volumetric and organic.

2-D objects take on 3-D behaviors

Although 2-D objects exist in NUIs, they can take on realistic 3-D behaviors borrowed from the real world. For example, a photo is flat but it can be bent and folded physically just as in the real world. The overall experience is physical and feels spatial as well. Such behaviors need not be rendered in true 3D to be successful; many 2.5D techniques can be utilized to maintain the illusion (and suspension of disbelief) of volumetric objects and behaviors.

Navigating through volume

In 1977, Charles and Ray Eames directed a short film titled [Powers of Ten](#).¹ The film began with a bird’s-eye view of a couple enjoying a picnic in a lakeside park. The camera started zooming out, moving an order of magnitude farther away every ten seconds, until it reached the edge of the universe. Then the camera started zooming back in until it reached the couple in the park. Finally, it started magnifying the scene by an order of magnitude every ten seconds, revealing a universe of molecules and atoms until it reached the nucleus of an atom.

This film changed the viewers’ perspectives of their surroundings. It provided a macro and micro perspective by revealing “content” as it zoomed in and out of the scene. Much as in *Powers of Ten*, navigating through a Surface experience is volumetric and spatial. The flat planes of a GUI window are extruded into a cube, which can represent a window, an application, a menu, or content. You can zoom out to see it in its entirety and turn it around to explore its surface and the space around it. You are invited to move through the environment and zoom in to experience the space inside from close-up, which becomes a space you can dwell in.

Using your spatial memory

Microsoft Surface experiences represent objects volumetrically and leverage spatial memory. Environments can extend well off-screen, and be dragged around to relocate content. Objects can be stacked in 3D space, using depth to sort, distribute, or establish focus upon content, As long as the user

¹ <http://www.youtube.com/watch?v=BBsOeLcUARw>

has a gestural means for navigating the environment and orienting themselves, they can create a mental model of the space, its content, and the gestures needed to access that content without needing to see it all on-screen. This use of volume and space can leverage the user's depth perception and their spatial memory. The user's association between what he or she wants to do (e.g., play a game) and where he or she does it (e.g., in a game application) develops naturally from memory-triggered context.

GUI vs. NUI WIMP (Windows, Icons, Menus, and Pointers)

Traditional GUIs include a set of standard controls and components that have been developed over time. They are known as Windows, Icons, Menus, and Pointers (WIMP) controls. In GUIs, they appear planar and flat. In NUIs, they are volumetric and leverage the spatial aspect described above.










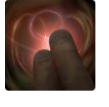
	 GUI	 NUI
Windows	Planar 	Volumetric 
Icons	Metaphors 	Metonyms 
Menus	Planar 	Spatial 
Pointers	Cursors 	Auras 

Figure 3.1: A comparative look at the old WIMP model in GUIs versus the new, spatial WIMP model in NUIs.

Windows

GUI windows are planar and appear in two dimensions as flat rectangles, though sometimes they appear in two-and-a-half dimensions as rectangles with shadows. They appear as a singular entity, and transitions between windows are rapid (windows appear and disappear, and it is not always clear which application launched a window).

In NUIs, windows can be thought of as the entire environment, which is larger than what can be viewed on the screen at any given time. They are connected to one another fluidly almost as if moving through a tunnel. Transitions between NUI windows are smooth, and it always clear where they lead and what lies behind them. You can pan across and zoom in on the environment in all three dimensions and rotate an object to affect change. You can enter and exit content by zooming in and out of the z-axis to reveal or hide information. The overall experience feels as if you are navigating through a world that is spatial and volumetric. There is a sense of infinite scaling and panning at the macro and micro levels. To perpetuate this sense, it is important to reduce (or even eliminate, where possible) the number of pop-up messages, modal dialogs, and other interruptions that can disrupt this sense of space and place.

Icons

GUI icons are metaphorical. They represent an action or a container of content (e.g., a folder). In NUIs, icons are not metaphors but *metonyms*. A metonym is one thing that refers to a more complex entity, such as the White House representing the U.S. government, the Hollywood sign representing the film industry, and “the press” representing all news media. Metaphors replace the thing they represent while metonyms represent the thing itself. For example, a metonym of a music album would not be a square icon with a picture in it or a music-note illustration, but rather an image of the album itself, in a form the user can recognize: a compact disc jewel case, with the album cover facing the user. It represents the musical content within the album, and is an interactive object itself; perhaps touching it will flip it over and let the user play the album (and, of course, CDs are not self-playable in real life). In NUIs, metonyms represent content and content is the interface.

Menus

GUI menus are linear in their behavior and planar in their form. While they sometimes exhibit shallow depth for clarity when overlapping other windows, they do not appear inherently volumetric or spatial. In NUIs, it is important to break down the concept of a GUI menu (or any other user interface control) to its original intent and inherent purpose, and then use volumetric and spatial principles to make them truly appropriate to the medium of surface computing. The next step is to merge these new forms with an appropriate metonym to create a control that could only exist in the NUI world...and that can only be as usable and natural on Surface.

Menus represent lists of mutually exclusive choices, some of which lead to other choices, and selection of one of these choices triggers an event of some sort. Adding a spatial aspect to NUI menus allows each item in a menu to be content itself, and for additional levels of menus to appear only when you need them. Rather than using “fly-out” submenus, users could stay more focused by drilling into subsequent layers of the menu itself. In the music album example above, a track listing on the back of a CD would act as a menu, on the “back” of a virtual CD jewel case. Perhaps, in addition, further playback controls appear within an item in that menu when it is touched, or additional information, such as options to buy the track being played (thereby acting as an inline sub-menu), could appear as if the menu item rotates in place to show a hidden face of a 3D geometry. NUI menus can and should be rich, volumetric, dimensional, contextual, and flexible.

Pointers

GUI pointers show the location of the cursor. In NUIs, the mouse is replaced with fingers and physical objects that directly interact with the experience, eliminating the need for a synthetic input device. Therefore, NUIs provide feedback called “auras.” Auras respond to user input, provide focus, and even provide clues for possible interactions. Every user input gesture should result in an aura of some kind, even if where the touch is registered doesn’t trigger an event, so that the user is assured that their input

has been “heard”...or, rather, that their touch has been felt by Surface.

NUIs must recognize a larger area than a GUI pointer, as well as opaque objects that are placed on the surface, such as fingers, hands, and physical objects of various sizes. Auras typically have soft or blurred edges to accommodate objects with imprecise boundaries or collections of objects. Auras tell you that your input has been registered. They can tell you what object of many has focus. An aura can fade over time, releasing focus automatically if the user ceases interacting with Surface (this action itself incenting the user to restart their interactions). An aura can even show a user where their next gesture should be placed, or how, based on the context of what they are trying to accomplish.

Think of auras as persistent feedback entities that make Surface feel responsive, aware, and spatial.

Guidelines

The following guidelines govern the design of experiences that adhere to the core idea of spatial.

- Make the structure of every WIMP element feel as if it has volume. The experience must feel exploratory and invite people to navigate through the volume as if it is their own world.
- Provide 3-D depth that enables people to zoom in, zoom out, pan, and change the orientation of the space.
- Leverage spatial memory where applicable and possible. Remember that the potential volume of interactive space can be larger than can be viewed on the Surface screen at any given moment. Allow people to understand that volume can be not only a vast two-dimensional canvas, but potentially also a fully 3-D volume in which content is located and activities occur.
- 2-D objects, such as flat photos, require additional characteristics and behaviors in order to fit well into a 3-D world. For example, when a photo collides with another object, its corner can curl up three dimensionally.

3.1.4 Superrealism

Because touch is inherently physical, it creates a sense of direct interaction with and control of technology. More fluid, natural experiences can be created not only by mimicking existing real-world physical interactions, but by augmenting them beyond what is possible in the real world. *Superrealism* pushes beyond what is physically natural so that experiences do more than what is possible in the real world. In this way, superrealism is natural and direct, but not purely literal.



Figure 3.2: Just like the real world, a user can touch a photo and push it around...



Figure 3.3: ...however, *unlike* the real world, a Surface experience allows a user to scale the photo with a two-finger gesture:

Natural interactions are created using gestures that people discover naturally because they match natural gestures in the real world. Superreal interactions are created by exceeding what is possible in the real world by leveraging the possibilities of virtual objects in digital environments.

Every action gives an immediate reaction

Just as in the real world, the nature of direct manipulation results in an immediate response to all actions. Every time a finger, hand or physical object interacts with a Surface experience, a response will occur making the experience feel “alive” and natural.

Natural and better than real

The content and objects in a Surface experience have a “better than real” quality that echoes the physical nature of real-world objects. Designing such an experience is particularly challenging because it is not about replicating natural objects and interactions; it is about making objects do more than what is possible in the real world. It imbues those objects with “superhero” capabilities that make characters such as Superman stand apart from the ordinary world because he has X-ray vision and superhuman strength. Superrealistic objects with superrealistic capabilities give people a sense of control not possible with other methods of interaction.

Guidelines

The following guidelines govern the design of experiences that are superrealistic:

- Make everything feel natural in its physical behavior. Push the boundaries of real-world metaphors and invent new and intuitive ways of directly manipulating content through touch and gesture. In doing so, be sure to support the users’ tasks and goals, and do not innovate for the sake of novelty.
- Make the experience feel organic and fluid. The “natural” feeling of Surface experiences is achieved by copying the organic movements of natural physical behaviors so that the experience behaves as naturally and instinctively as the real world. It feels like second nature and leverages the unique possibilities enabled by the physical interaction. Each experience feels smooth, organic and fluid even when the interaction results in superhuman capabilities.
- Physical objects should interact naturally. Mimic and augment natural physical behaviors through the gestures that a person performs with a physical object and through the way that the Surface experience reacts to the physical object.

3.1.5 Contextual Environments

The Surface experience tunes itself to what people are trying to do, where they are, and why. It reacts

based on the context of the internal and the external environment.

Internal Environment

The internal environment refers to the experience that happens inside the interface. The Surface experience responds to the people and objects that touch it. It reacts and adapts based on the context that a person is in at any given moment. In other words, Surface experiences respond to people's actions based on where they are in the experience and their behaviors. In the game industry, this is known as causality, which describes the relationship between action and consequence. In a game, causality must be obvious at all times. Players must understand that what happens is a direct result of their actions and that through learning they will be able to anticipate and control the consequences of any action. They are able to identify the rules of cause and effect. If they did not, the game would feel random and their ability to effect change in the world would be stymied. Microsoft Surface experiences respond to people's actions in a way that clearly results from their context and behaviors.

External Environment

The external environment refers to the physical environment in which the Surface unit is located. The experience is influenced by its physical surroundings, such as the location and audience, the lighting, the objects, and the nature of the experience itself. For example, cultural differences in audience will significantly impact how closely clustered seating might be, and the seating density will also influence how Surface will be used. Whether or not there is a change in pattern or material, internal décor can impact how people in a venue might be attracted or drawn towards having a Surface experience. If a Surface experience is centered on placing certain objects on a Surface screen, where those objects are made available and where they are returned when not in use on the screen can impact the discoverability of such functionality, or even in understanding the overall goals of the experience. The context of the objects and experience in the physical environment leads the Surface experience to feel like part of a larger ecosystem.

Subtle Affordances

Subtle affordances invite people to interact with the experience even when interaction is not required. They invite people to discover new content and functionality. They are not intended to be explicit commands or affordances that people must take action on in order to move through the experience, and they are not necessary to completing a task. They are highly visual and behavioral elements that are integrated tightly into the design and support the design concept or metaphor. For example, suppose you are looking at photos spilled onto the Surface screen and a few of them include annotations. The corners of the annotated photos could curl subtly to provide a visual and behavioral cue that invites you to interact with it and discover the message attached to it. Touchable edges, corners, and other areas of virtual objects could even be subject to virtual wear and tear, so that aging or weathering effects could imply where touches will be effective...and even could increase over time as the objects or regions are touched over and over again. Subtle affordances can be discovered to create a rich experience that is contextual to what a person is doing in the experience.

Guidelines

The following guideline governs the design of contextual environments.

- Take both the internal and external context into account. Aim to understand the context of people as they interact with the experience, and then scope the design solution to emphasize simplicity over complexity. To accomplish this, examine people in the context in which they are interacting, and create Surface experiences that dynamically adapt based on this context.

- Consider the physical environment in which the Surface unit is located, its physical properties, and the objects that surround it. This will impact how approachable and welcoming the Surface experience can be, as well as helping to build awareness around the presence of Surface in a venue.
- Lead by example with subtle affordances. Simple, focused visual cues will help users anticipate what gestures they should use, and where, and what the results might be. The experience can be effortless to learn because it inherently feels familiar.

3.1.6 Scaffolding

In the constructivist theory of education, scaffolding was originated by psychologist Lev Vygotsky (Berk, Laura E. & Winsler, A. (1995). *Scaffolding Children's Learning: Vygotsky and Early Childhood Education*. Washington, DC: National Association for the Education of Young Children). Scaffolding is a teaching method that breaks down bigger challenges ("How does this whole system work?") and focuses on smaller problem-solving challenges ("How do I initiate this one action?") through specific prompts, hints, and leading questions. It is a process of supportive structures and moments that encourage active exploration instead of memorization and repetitions. The pleasure in finding the solution is in the journey.

Scaffolding can be an effective method in structuring Surface experiences. This can be done by optimizing tasks into small, self-evident steps with maximum visual or emotional reward, and by simplifying moment-to-moment decision-making.

Exploration and Linear Experiences

There are two ways to move through a typical computing experience. The first is the *exploratory experience*, which allows a person to discover new content and functionality through exploration, often without a specific goal in mind (e.g., browsing books, "window shopping" for clothes, reading interesting articles, playing games, etc.). The second is the *linear experience*, which guides a person with a specific goal in mind from point A to point B (e.g., checking out an online purchase, printing directions, etc.) Both of these modes are available in GUIs and NUIs, but with radically dissimilar end-user results.

Both exploratory and linear experiences in GUIs tend to offer a high number of paths depending on the context, application, and intent. In order to accomplish a set of tasks, the user must find the individual places in the UI that allow each command to be run. For example, copying files would take place in the application shell, finding the local time and weather would require finding and launching a browser and querying a search engine, and checking the integrity of a hard drive would require finding and launching the right application.

On the other hand, the number of features and available choices in Surface experiences are significantly lower than typical WIMP paradigms, while the few choices the user has are richer, more fun, and more rewarding. For example, there may be only one path to print (no right-click menus, no complicated decisions about where and how to select a printer), but perhaps that path involves literally throwing photos onto a visual representation of a printer, putting people at ease and providing a fun and visually rewarding experience for an otherwise mundane task.

In other words, superfluous features and paths have been eliminated in favor of providing one simple but compelling solution. The experience is designed to guide people to a solution. It becomes the "scaffolding" that supports the discovery of a solution by offering a rich and pleasurable journey to reach it.

Reducing Choice Paralysis

Simplicity is a key element to effective Surface experience design. When there are too many choices to consider, the analysis process grows too complex and the user starts to feel confused and overwhelmed (Schwartz, Barry (2005), *The Paradox of Choice*, Harper Collins). Users should instead be presented with only the fewest reasonable choices at a given moment. This simplifies decision-making and sequences the disclosure of information or required choices over time, increasing a sense of simplicity of thought and action, and therefore ease of use and ultimately enjoyment. Simplicity need never mean simplistic; simple processes and tasks can be incredibly rich and powerful. Microsoft Surface experiences should prove that robust results never require complex procedures.

Guidelines

The following guidelines govern the design of experiences with few paths.

- Encourage the discovery of new content through exploration. The more a person explores, the more content and functionality are revealed.
- Reduce the number of available paths and choices. Applications with a high number of complex options and features are not ideal candidates for a Surface experience. Microsoft Surface experiences focus on the most important tasks by removing or limiting irrelevant tasks and features. Create visual experiences that match the expected behavior.
- Make each element visually appealing. When an experience offers people fewer elements to interact with, every element should be visually appealing and beautiful, and have a high degree of craft. Remove excessive numbers of choices and enhance the details to create a rewarding and magical experience that makes people feel as if they are experiencing something of the utmost quality.

3.1.7 Performance Aesthetics

In video game design, beauty can be described as that which pleases the senses. The classic game *Myst* and its sequel *Riven* are “beautiful” adventures where the primary motivation outside the narrative is simply to explore the beauty of the world that had been created. In this example, aesthetic beauty is more about the recognition of patterns than in discovering new ones. It gives people a sense of delight when they recognize these patterns but are slightly surprised by some nuance of them. The nature of beauty, however, is somewhat fleeting and recognizing it is a highly subjective response. The challenge is to create something beautiful enough to stand out in a visually saturated culture.

In addition to visual beauty, the aesthetic of performance applies to Surface experiences. The physical nature of touch, gesture, and direct manipulation requires seamlessness between objects touching the surface and information appearing on the screen. The aesthetic demands no performance lags or hiccups that reveal those seams and break the suspension of disbelief. Game engineers go to enormous lengths to create high-performance experiences, removing delays in loading and any other operation that might cause an interruption in visual fidelity, frame rate, or acoustic fidelity. Microsoft Surface experiences should be just as flawless, responsive, and immediate. The accuracy and smoothness of the performance are critical to an experience that feels like second nature. High performance itself creates beauty.

Deriving pleasure by doing

In Surface experiences, people engage in multi-sensory play that is fun and pleasurable. The Surface experience creates an emotionally engaging experience as a result of forethought rather than as an

afterthought. One can think of these emotionally engaging experiences as playful. “Play” is often miscast as the antonym of “work” and therefore is cast as a whimsical, time-wasting activity. However, it has enormous value in enabling participants to imagine, discover, and experiment with the world as well as establish important social boundaries, relationships, and rules. There are a number of aspects that make Surface experiences playful.

That which pleases the senses

The first two senses that Surface experiences appeal to are *sight* and *sound*. Microsoft Surface experiences focus on highly crafted visual representations of content, motion, and sound that please the senses.

A third sense is *imagination*. The discovery of new places, things, or effects rewards players by stimulating their imagination with visually compelling cues.

A fourth sense is *challenge*. Finding solutions to these cues are challenging but not daunting problems to solve. In the game industry, serious gamers are drawn to cues or “hooks” that provide an adrenaline rush, such as frenetic action, time limits, and threats to safety that artificially enhance the pressure to succeed.

A fifth sense is *pacing*. Successful game play and pleasing the senses can be boiled down to well-paced decision-making. Each player has a set of rules, a context, and a goal. It is left to the player to find the most effective way to reach the goal given the context while adhering to the rules. The last sense is immersion. Immersion refers to the state of being deeply engaged or involved. An “immersive experience” involves being in a different environment reached by physical means or by the use of one’s imagination. The pleasure of “escaping” to this environment stems from exploring its systems, rules, culture, and space.

Mihaly Csikszentmihályi, the author of *Flow: The Psychology of Optimal Experience* (1991, Harper Perennial), illustrates his theory that people are happiest and the most fulfilled when they are in a state of “flow;” in other words, when they are experiencing a feeling of being “in the zone or groove.” It is a state of absolute concentration on a task. Microsoft Surface experiences invite people to feel so fully engaged that they temporarily forget about their daily activities and lose their sense of time. To achieve this state, their task cannot be too easy or too difficult. Their sense of fun is derived from focusing less on the immediate goal and more on the journey to reach it.

Guidelines

The following guidelines govern the design of experiences with performance aesthetics.

- Make the experience responsive. All actions should have immediate and obvious consequences. The “wait cursor” is an artifact of the past and should be removed from the library of software experience design. Good experiences do not make people wait for even a moment because it breaks their immersion, flow, and concentration.
- Strike the right balance between being neither too easy nor too complicated so that the experience is engaging, immersive, and fun. Puzzle games such as Tetris achieve this balance. It challenges players to recognize a shape and quickly manipulate it into a gap. More complex problem-solving games require players to use their creativity to solve the challenges presented. Figuring out a way to do something is always rewarding, but figuring out a unique way to do it yields even greater feelings of reward.
- Give everything the utmost quality. Microsoft Surface experiences focus on highly crafted visual representations of content, motion, and sound to help create fun and pleasurable experiences.
- Make the technical performance flawless. The experience demands a flawless running that removes any delays or hiccups in performance.

- Make the experience “autotellic,” a term coined by Mihaly Csikszentmihalyi to describe the ability to derive optimal experiences through the pursuit of activities that are performed purely for the intrinsic benefits of performing them. In other words, pleasure is derived from the journey as opposed to the outcome. In Surface experiences, pleasure is also derived from the behavior and interaction rather than just the accomplishment of a task.
- Understand the technical constraints. Performance is only as good as the slowest element in the experience. For example, a wireless connection is constrained by the through-put of the connection. If an experience requires the connection to be faster than your connection allows, then the experience will be negatively affected and the flawlessness of the performance will be broken.

3.1.8 Direct Manipulation

The Surface experience responds to fingers and objects by encouraging touch and direct manipulation, whether by one or many fingers or physical objects. For example, explore a music library with friends by flipping through album covers, or play a chess game by using physical game pieces. The actions become temporal and unmediated in the way that content is manipulated.

Discrete to continuous

Touch, gesture, and direct manipulation in Surface experiences move away from discrete actions toward continuous ones. In GUIs, discrete actions are mostly brief, single-click actions performed in a sequence to complete a task.

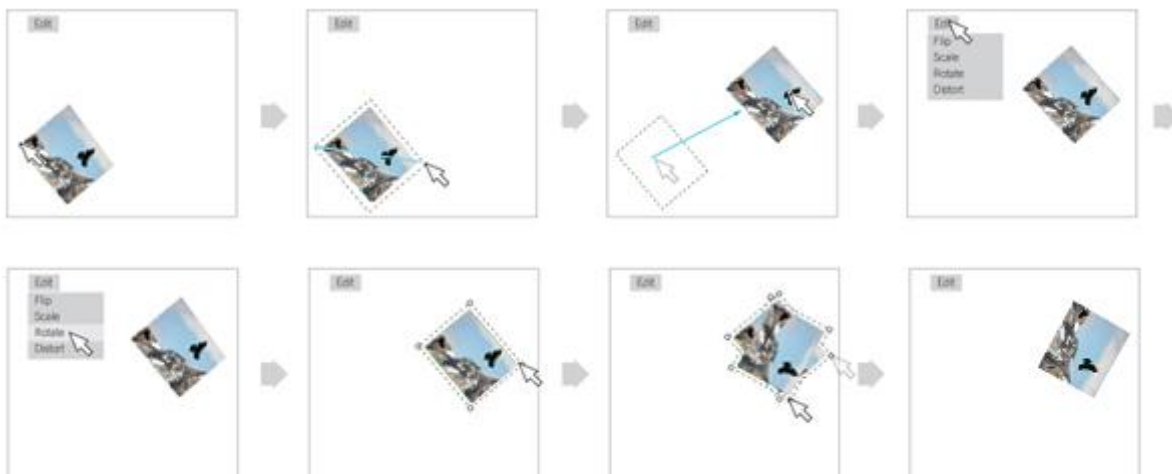


Figure 3.4: To move an object from one location to another in a GUI, select the object, select the appropriate command, and then move the object.

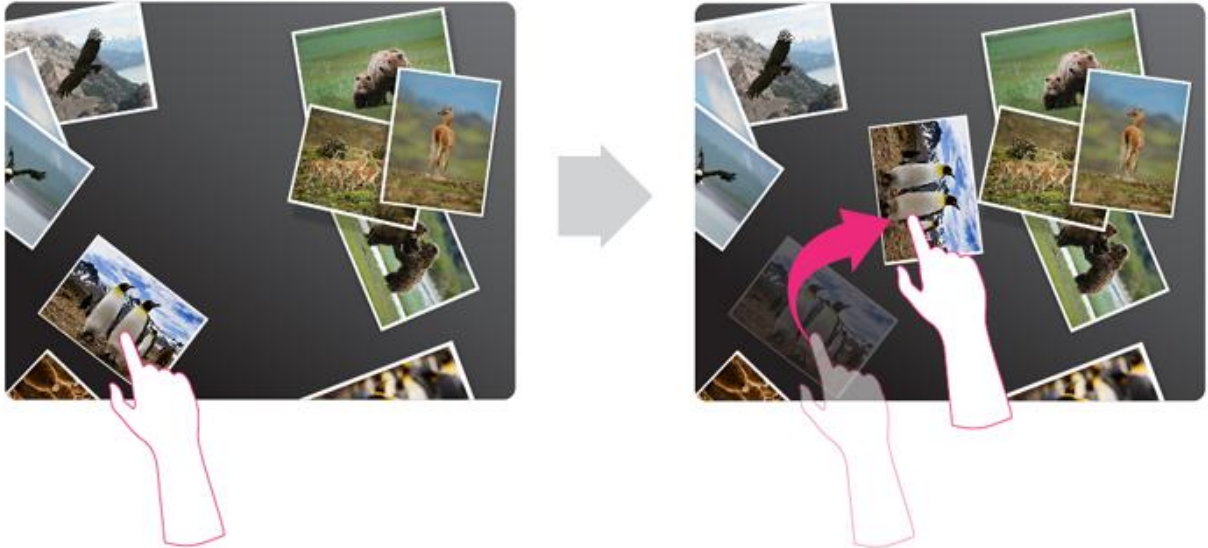


Figure 3.5: Direct manipulation favors continuous actions. To move an object from one location to another, simply grab it and move it to its new location.

Low Learnability

The nature of direct manipulation results in natural interactions that require little or no learnability. The learnability of GUIs is often very high. For example, right-clicking or using the ALT and CTRL keys are mechanisms that people need to learn how and when to use. In Surface experiences, touch and gestures are simple and intuitive enough so that the learnability is low and can sometimes occur through discovery. For example, you can move a photo naturally by touching and pushing it around and in doing so, you might discover that you can scale it by pinching it.

Guidelines

The following guidelines govern the design of experiences that are optimized for direct manipulation.

- Leverage the fact that people have many fingers. Many fingers and many physical objects can interact with the Surface unit at the same time and should be encouraged through the design. The dynamic nature of multi-touch requires a willingness to challenge the status quo and push the boundaries of creativity for design solutions aimed toward many people and many objects.
- Remember that size matters. In GUIs, the position of the mouse is represented as a single point on the screen. With fingers and objects being used as input devices, interactive elements must be properly sized to accommodate these input methods and be positioned so that the user's hand, arm, and/or input object don't block relevant content surrounding an interactive element.
- Expand the new vernacular that direct manipulation inspires through touch and gesture; tap, flick, squeeze, pinch, push, scoop, slide, etc. Push the boundaries of what is possible and what feels natural and intuitive.

3.2 Key User Experience Benefits

Every designer, developer, or manager involved in creating Surface experiences can apply some simple heuristics to his or her work to assure that the experience is optimized for natural user interfaces and the surface computing paradigm as a whole. This section contains some basic guidelines that summarize the

Surface philosophy and provide a framework for determining the overall success of a Surface user experience. These benefits should be kept in mind during every phase of a project, and during every discussion around features and functionality, from the most significant to the most subtle.

3.2.1 Microsoft Surface Is Natural and More Intuitive

Users already know how to interact with Surface. There is nothing to learn: everybody already knows how to touch, and touch is all anyone needs. The items seen on the screen are familiar objects; the interactions are immediately learnable. Applications are easy-to-use as features are discovered in the first few minutes of exploration, yet those features continue to evolve and keep the users' interest. Appearance and animations are soft, organic, and free-form. The technology is invisible; there is no technical jargon, there are no technical tasks, and there are no expectations of technical expertise and knowledge. Users fall into the flow of the experience without having to think about how to use the technology.

Microsoft Surface is:	Microsoft Surface is not:	<i>Desired responses:</i>
Familiar	Stale	"I did not have to think about how to do it."
Simple	Simplistic	"I did not need instructions."
Obvious	Patronizing	"It does not feel like technology."
Easy-to-use	Shallow	"Finally someone got it right."
Like the real world	Literal	"I will never use a mouse again."

3.2.2 Microsoft Surface Is Unique and Magical

Users feel a sense of wonder. They are quickly immersed in a familiar experience that is also unexpected and surprising. Powerful subtleties emerge that draw users in and spark their imagination. Time is forgotten as play and discovery take hold. The ordinary and mundane become extraordinary.

Microsoft Surface is:	Microsoft Surface is not:	<i>Desired responses:</i>
Extraordinary	Gratuitous	"How did they do that?"
Unexpected	Unwelcomed	"That is amazing!"
Delightful	Distracting	"You will not believe this."
Inspiring	Superficial	"That took my breath away."
Powerful	Overwhelming	"Wow!"
Compelling	Superfluous	
Better than real	Unnatural	

3.2.3 Microsoft Surface Is Premium and Authentic

The Surface experience exudes quality, meaning, and delight. The sense of simple elegance creates desire. The experience reflects thoughtful attention to craft and quality at every level of detail. The experience is sensual, connecting on an emotional level. Users are touched beyond an immediate attraction to

something new or trendy.

Microsoft Surface is:	Microsoft Surface is not:	<i>Desired responses:</i>
Well-crafted	Cookie-cutter	"I want to show it off."
Innovative	Blue sky	"They paid attention to the details."
Genuine	Forced	"It has all the right stuff but nothing more."
High-end	High-brow	"It just feels right."
Gorgeous	Pretentious	"I wish everything was designed this way."
Smooth and continuous	Abrupt or disconnected	

3.2.4 Microsoft Surface Is Aware and Responsive

Microsoft Surface is always ready. It does the right thing at the right time in the right way. Interaction with Surface is fast, responsive, and precise. Every user action creates an enhanced and appropriate reaction, transforming even ordinary tasks into efficient delightful gestures. Microsoft Surface feels alive, as if it were an extension of a user's will, yet it anticipates and interprets along the way. The sights, sounds, and reactions of Surface give the user a sense of having direct influence over something that has an energy of its own.

Microsoft Surface is:	Microsoft Surface is not:	<i>Desired responses:</i>
Efficient	Dull	"It seems like it is alive."
Alive	Human	"There were no barriers."
Intelligent	Presumptuous	"It did what I wanted it to."
Anticipatory	Didactic	
Flexible	Chaotic	
Immediate	Hyperactive	

3.2.5 Microsoft Surface Is Social and Collaborative

Users enjoy having a Surface experience together. When they do, they find themselves sharing, collaborating, playing, and competing.

Microsoft Surface is:	Microsoft Surface is not:	<i>Desired responses:</i>
Collaborative	Selfish	"Join me."
Open	Controlling	"We get more out of it together."
Inviting	Exclusive	"We are face-to-face at last."
Flexible	Linear	
Social	Isolated	

4.0 User Interface Overview



4.1 Putting the Philosophy and Principles into Practice

Microsoft Surface plays a significant role in completely redefining existing human-computer experiences through touch and gesture. As illustrated in the Surface experience philosophy (see [Vision and Philosophy](#)), designing for these experiences is not about retrofitting existing interaction models and metaphors. It is about applying refined methods that consider the full spectrum of what is possible with touch, gesture, and direct manipulation.

Designing for touch and gesture is not about doing old things in new ways; it is about doing new things naturally.

This section outlines the design considerations and implications to the design process that should be adhered to in order to design successful Surface experiences.

4.2 Considerations

This section is intended to frame a number of diverse areas that have an impact on user experience.

4.2.1 Project Considerations

Every design and development project is unique and the resulting output is greatly affected by scope, resources, process, and time. Why talk about these things in a document focused on design guidelines? It turns out that the project realities of skill sets, process, and time end up being the factors that most significantly impact design effort and overall solution quality. Therefore, this small section is devoted to providing some insights based on real experiences. It is our hope that these guidelines, combined with excellent resources, a good design process, and a realistic timeline, will yield great results.

Design Skills

Start with the proper people, who have the right skills, and they can inform and drive a timely process to deliver a quality solution. Solution development teams should have a multidisciplinary core team in place early to help inform and drive key decisions around scope, time (schedule), and process. Designers are frequently stereotyped as visual or graphic designers, who are brought into a project toward the end to help apply cosmetics and a beauty treatment. It is very important to have designers be a part of the core team from the beginning. There are different types of design skills to consider:

Designers	Deliverables	Project Phase
Interaction Designer	<ul style="list-style-type: none">User-system model defining the flow and behavior of the system (scenarios, flow diagrams, storyboards, wireframes, prototypes, etc.).	Critical core team member during pre-production and production. The most important work happens before development starts, during pre-production.

Visual Designer	<ul style="list-style-type: none"> • Visual design language (artwork and animations) used to convey functionality and brand. • Visualizations and concept sketches during pre-production to bring everyone onto the same page. 	Important core team member during pre-production and critical during production.
Technical Artist	<ul style="list-style-type: none"> • UI XAML programming (for Microsoft Windows® Presentation Foundation [WPF] applications) and insight into design feasibility, edge-case scenarios, and costs (i.e., time). • Proof of concepts, prototyping and development. • A more polished UI by operating between the designers and the developers to create better efficiency. 	Important core team member during pre-production and critical during production. Early collaboration with a Visual Designer would allow for a more real-time exploration of look and feel, helping the team arrive at a visual solution faster. Early collaboration with Interaction Designers enables effective usage and customization of UI controls.
Audio Designer	<ul style="list-style-type: none"> • Audio design used to provide necessary UI feedback and ambience that complements both the interaction and visual design. 	The nature of the application in question affects the importance and prevalence of audio design (e.g. games vs. productivity application), which in turn can drive your investment decisions.

It is always possible you may find a single resource with multiple skills; however, having specialists instead of generalists will improve both the overall quality and production efficiency.

Scope

Microsoft Surface experiences trim features and focus on making the few options available immensely rich. Consider narrowing the number of choices to the bare essentials and focus your design energy on making them the most rich and rewarding experiences possible.

Whatever you chose to do, do it well or not at all.

There is always a temptation to push for more features and functions, but we highly recommend adopting a “less is more” strategy. When in doubt, leave it out. Refuse to add features that detract from the clarity and simplicity of interaction. Ask yourself:

- Does every feature contribute to a single overall purpose of the application?
- Do most users encounter all of the features in their first five minutes of use?
- Does each feature contribute something special that is worth its added complexity?

Time

Most projects start with goals and intentions of producing something great. Too often, those goals are quickly forgotten and replaced by a drive to meet a schedule. In the end, members of your team may understand why something came out compromised and how it will be fixed in the next version, but the users neither know nor care and will not be so understanding or forgiving.

In our experience, we continually see project teams underestimating the amount of time it will take to

fully design and develop a feature. A high level design is created up front for the sake of estimating a project schedule, but later when detailed design is done on a specific feature, the team realizes that it will include ideas, animations, capabilities, and much more that the project simply cannot and did not take into account. This forces in situ decisions to either scope the feature to the time allotted, or change the schedule. Ultimately, it is best to ensure that there is plenty of time built into the schedule for each feature, avoid creating schedules that are too tight and have things “squeezed in”, and be prepared to bump off lower priority features in favor of better designing and developing the desired features.

Process

A different section (see [Methods and Tools](#)) deals with process in more depth, but we will briefly mention it here since it is a reality of a successful project.

The greater the investment in pre-production, the greater the payoff during production.

Do not skip or short-change the pre-production phase of a process. Build more time than normal into pre-production even though the exploration and definition activities may not seem to be as tangible as real code. Insist on market research, user studies, and iterative prototyping that transitions from low fidelity paper mockups to high fidelity versions created in the same technologies used for the final product² (e.g., WPF or Microsoft XNA®). The value of the real code depends on these pre-production investments. First-timers should give themselves much more time than normal to climb a new learning curve and to iterate many times before designs are considered final. As easy as it may be to understand and value Surface’s unique attributes, creating design solutions that effectively use them is not as easy.

Lastly, build more time than normal into the back end testing phases of your process. The extra time is needed not only for finding bugs (technical, performance, and design), but to also address them and ensure that a premium, quality solution is delivered.

4.2.2 What Qualifies as a Good Microsoft Surface Application?

Obviously, just about any application can be ported to or created for the Surface platform. Furthermore, there is rarely a shortage of ideas. However, having applications representing the right quality of experience is more important than having a high quantity of applications. Microsoft Surface experiences should exude quality, meaning, and delight. The application reflects thoughtful attention to craft and quality at every level of detail, touching users more deeply than their interest in “what’s new or what’s trendy.” All of this then begs the question, are there some criteria or considerations for determining the fit and appropriateness of an idea or application for Surface?

When selecting or prioritizing possible applications, consider asking the following questions about them:

Questions	Impact	Notes
Is it useful, relevant, and rewarding?	+++	For its intended audience, the application must serve a purpose or address pain-points. Furthermore, give users a sense of enjoyment and accomplishment, as well as provide a valuable experience to entice repeat visits.

² Team members will have varying degrees of comfort and expertise with different prototyping tools. By ultimately using the “final” technology (i.e., WPF or XNA), teams avoid the need for last minute design changes during production due to prototyping tool incompatibilities, limitations, or differences.

Questions	Impact	Notes
Is it naturally multi-user / social?	+++	Social is a key Surface design principle and attribute that differentiates this platform, providing opportunities for great experiences that users desire and expect. Social imposes requirements on both the software application design and physical micro-environment around Surface.
Does it benefit greatly from or rely on multi-touch? Is it naturally gestural?	+++	Microsoft Surface is not a single-touch kiosk. Avoid applications designed so that everything can be accomplished with single touch.
Does it benefit from object interaction?	+++	One of the primary benefits and capabilities of Surface is to blend the boundaries of the physical and virtual worlds.
Is it non-directional (i.e. no "up" direction)?	++	Create a 360 degree UI and ensure users can approach and use the application from any side.
Is it "walk-up-and-use" easy?	++	For commercial environments in which applications are used by the general public, users must be able to simply walk up and start using the system with none to very minimal perceived start-up cost (e.g., unreasonable learning curves, lengthy instructions, tedious pre-conditions or setup). The sense of simple, elegant quality creates a "must-have" desire.
Does it align and support business success criteria?	++	Obviously, no application will be created and deployed without reason or specific objectives. However, circling back to ensure the application's key benefits and purpose align with business metrics may inform application selection and prioritization.
Can it be adapted for Surface if it already exists on another platform (i.e., PC or web port)?	++	Of course, just about all applications can be technically ported, so this question is more about adapting the heart and soul of the application, and rethinking and redesigning the approach and interaction model. Embellishing a GUI application by trying to accommodate multi-touch is not a good strategy and will not lead to NUI. Ultimately, if the application can be used just as effectively on a standard PC or touch screen, then there may be no need for it to be a Surface application.
Is it appropriate for use in a public space?	++	If the application will be used in a public space, consider any possible privacy / security concerns and ensure the ability to deal sensitively with personal information.
Can content itself be the primary means of interacting with the application?	++	The content is the interface. Users directly move objects on the screen. Controls may be attached to the content, but the non-content user interface is limited. Controls are lightweight and are relevant and necessary for the content. At all times the application preserves the illusion that users are interacting directly with the content itself.

Questions	Impact	Notes
Does it have minimal manual data entry, if any?	+	Occasional and short data entry is acceptable, but any virtual keyboard can, over long usage, become frustrating in comparison to physical keyboard experiences.
Does it have minimal input and manipulation precision?	+	Microsoft Surface is not designed to replace high resolution mice or pointing devices needed for high precision actions or tasks.
Is it independent of "techno-legacy"?	+	When possible, find creative alternatives to using "old" technology (e.g., cables, magnetic stripe readers, media card readers) in order to keep technology invisible, interactions easy, and retain a sense of magic and wonder. Users should respond, "How did they do that?"
Can it quickly end at any point in time?	+	In commercial settings, fully expect users to simply get up and walk away. At best, they may explicitly instruct the system to reset and expect everything to close without a series of follow-up questions and other tasks or activities.
Is it participatory, engaging, fun, visceral?	+	Make the activity prized and create a fast and responsive interaction with the application. Users should be quickly immersed, finding the Surface experience unexpected and surprising. The interface and the technology are invisible; they just work. Users should fall into the flow of the experience without having to think about how to use the interface or the technology.
How much continuous interaction time does it require?	+	For best possible ergonomic experience (when the unit height is not customized/adjusted), consider applications that include brief, light weight tasks, and allow users to determine the length of interaction with Surface. Ensure there is plenty of adjustable seating and space for users to place legs and feet between Surface and seating.
Is it user controlled and event driven?	+	
Is it light on text?	+	Pictures, videos, and multi-media in general are much more engaging.

4.2.3 Social

Scenarios

Users want to interact with Surface together. When they do, they find themselves sharing, collaborating, playing, and competing. As such, "social" is one of our key design principles and is often thought of as being one and the same as "multi-user". Of course, having multiple users around Surface does not automatically make it social in the intended way. Before diving into various design ideas and considerations for social, first consider the various types of multi-user situations. Thinking about these different situations will help you better define your goals and design strategies:

- Location:
 - At the same Surface: Everyone is physically co-located around the same tabletop.

- At the same venue: Everyone is physically co-located at the same general area, possibly within listening or viewing distance of one another (e.g., restaurant, bar, lounge).
- Dispersed: Everyone is geographically dispersed (e.g., different cities).
- Connection:
 - Microsoft Surface units are stand-alone islands and disconnected from each other.
 - Microsoft Surface units are networked.
- Scenario:
 - Multi-stranger: Participants do not know each other.
 - Multi-friends: Participants do know one another.
 - Driver with passengers: One user drives and others participate peripherally.
 - Turn-taking: Each user takes a turn at being the driver; usually seen in games like poker, bowling, etc.
 - Mixed: Multi-friends are at one Surface unit, but networked to other units occupied by strangers, for example.

While these considerations are not necessarily comprehensive, hopefully they entice thinking about opportunities, goals, and best practices. For example, a Surface unit in a public space encouraging multi-stranger participation will impose social design strategies and considerations that are unique and different from a multi-friend scenario. These differing scenarios should impact ideas, features, and plans along the lines of continuity, resets, privacy, cooperation, independence, interference, etiquette, physical environment design, and much more.

See the [Interaction Design](#) section for a set of social design considerations.

Design Ideas

- Create cooperative interactions. Designing opportunities for many users to work at the same time toward the same end is an objective of good Surface experiences. Establish some simple ways in which the experience gets “richer” as more users participate at once, or as different users with complementary abilities affect the interaction. A simple example is to play a beat-matching game such as Rock Band where each user is assigned a different instrument. With more users, many more instruments can be played simultaneously, thus creating a richer experience. Another more organic example might assign different controls to different users (e.g., one user controls camera orientation while another controls movement within a virtual space). This allows multiple users to create an experience that a single user might not be able to experience.
- Reward intentional and unintentional collaboration. Let new things become possible when users collaborate; allow surprising results to be achieved when more than one user touches Surface at the same time. Can users achieve different results by collaborating? Does the application sometimes give surprising results when two users touch at the same time?
- Create community interactions. An example of cooperation that does not occur in real time is the simple ability to leave recommendations of places, events, or things to do on a virtual mapping application. This organic community of comments, stories, and contributions can have a deep impact on the design of the interaction.
- Consider designing competitive interactions. The joy of winning comes in part from seeing your

opponent's reaction to defeat. Competition is also a strong motivator for some personality types. Consider networking groups of Surface computers together in social environments to enable competitive activities to be played. They might take the explicit form of games (trivial challenges, card games, simple gambling, etc.) or they might be employed as a hook to engage users ("Can you design a better shoe than any of these others?").

- Consider designing experiences that facilitate a shared experience between users in different locations. For example, a user in New York can play chess against someone in Tokyo.
- Consider asynchronous social experiences. Facilitate a shared experience that invites users to work or play together at different times. For example, a user starts to draw a picture and walks away, and someone else continues drawing where the first user left off.
- Consider extending the capabilities of Surface experiences by leveraging social networks to collaboratively filter a set of results.
- Consider using social networks to provide scaffolding. For example, the Microsoft Xbox LIVE® online game service allows communities to help users become self-directed in the ways that they connect to their network and content.
- Consider manipulating content over distances. For example, if one Surface unit is connected to another, allow it (the relevant parts) to be manipulated remotely.

4.2.4 Object and Device Interaction

Microsoft Surface has an incredibly unique capability to "see" things that come into contact with its screen and reflect infrared light. This is how Surface is able to respond to touch. This capability extends beyond fingers and hands to objects, tags, and tagged objects. The result is a great opportunity for application designers and developers to introduce rather unique interaction possibilities.

Object Interaction

Imagine playing a game wherein the map is created using everyday objects simply placed on the Surface screen. For example, think of a software pinball game, except the objects off of which the virtual pinball bounces are real physical things (e.g., salt shaker, napkin, wallet, cell phone, or business card). As long as these objects reflect infrared light, Surface can detect their presence when they come into contact with the screen and the application can react however it was programmed to respond. This has several implications:

- Microsoft Surface will be used as an actual table. Users will place drinks, plates, handbags, or anything they might normally put on a table, onto Surface. Sometimes these objects will be off to the side, but when they come into contact with the screen the application can either ignore them or show awareness and respond. The response could be intelligent. For example, information on the screen could potentially collide with these stationary objects or avoid appearing underneath them. The response could be playful and take advantage of introducing some unexpected and delightful reactions.



Figure4.1: Microsoft Surface used as an actual table with many expected items placed on it.

- Users can employ real physical objects during interactions. For example, a paintbrush (with white colored bristles) could be used instead of a finger.
- There is a difference between detecting the presence of something (object sensing) and knowing or recognizing what it is (object recognition). Since these objects are not tagged, there is no guaranteed way of identifying or recognizing them. While it is possible to take camera data and try to recognize certain physical objects (e.g., based on size and shape), it is generally not feasible due to high costs and questionable results. If it is important for your application to recognize an object, then consider using Byte Tags.

Tags and Tagged Object Interaction

A Byte Tag is a geometrical arrangement of shapes that defines a value that the Surface Vision System can recognize. Similar in many ways to a barcode, these Byte Tags can be printed and attached to physical objects, making them tagged objects.

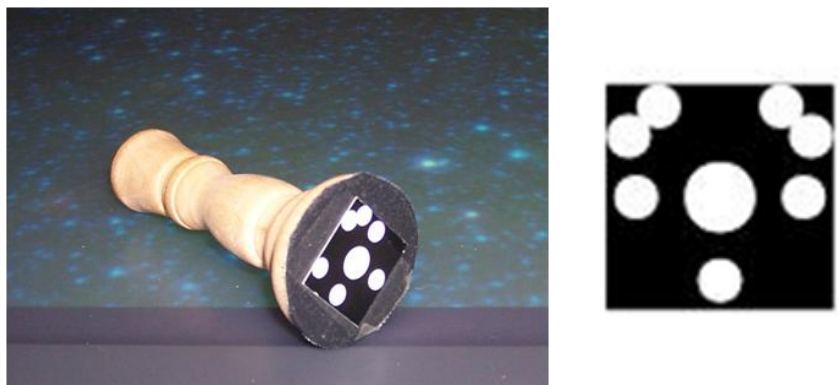


Figure 4.2: Chess piece with a printed Byte Tag underneath.

Surface applications can be programmed to recognize a physical object that is Byte Tagged and respond accordingly. Leveraging this capability creates richer and more seamless experiences between physical objects and the Surface interface. Some examples include a glass tile that acts as a puzzle piece in a puzzle application, a drinking glass that spawns a virtual coaster (complete with drink info, ads, or playful bubbles), and game pieces for chess.

Tagged objects can be placed on the Surface screen and moved. When moved, the tagged object can be picked up and placed back down on the screen, or it can be dragged over the screen never losing contact. In the latter case, the application “sees” the tag continuously and maintains recognition and tracking. In the former case, the application loses track of the tag and will not know if the object has been removed permanently or temporarily, and if it is placed down on the screen again, it will trigger the recognition process again. The application should have the relevant logic to determine how to respond with each pick up and placement, taking into consideration:

- Are events from previous placements still in effect?
- Is information from previous placements still visible?
- Does the application’s current context change anything?

In some cases, events and information should persist after removing a tagged object, and in other cases they should be tightly coupled and appear and disappear along with presence of the tagged object.

Tagged objects placed on the Surface screen can be rotated. Surface Byte Tags provide orientation information so users can rotate a tagged object and the application will detect this and respond accordingly. For example, a virtual interface presented along with a tagged object can be manipulated in response to manipulation of the tagged object. Thus, rotating a tagged object could cause the virtual interface associated with it to also rotate. This feature can be used in clever ways to reinforce a social interface. Keep in mind, however, that in many cases it is obviously important for the object and its associated virtual interface elements to be oriented in the same direction. This further maintains and reinforces the illusion of the physical world merging with the virtual, for if they do not act in concert the illusion is broken.



Figure 2.3: Example of glasses with Byte Tags and their associated virtual interface.

Other considerations include:

- Pieces can get lost. This is a simple reality when tagged objects are made available to the general public. In such cases, ensure the application is engaging and effective even without the use of tagged objects, thus avoiding any critical dependence. Additionally, less expensive tagged objects are more easily and cost effectively replaced.
- Even under venue staff control, do not assume tagged objects can be used in secure ways (e.g., to unlock or access private features or information). Currently, Surface Byte Tags only provide a total of 256 unique IDs, and the visible black and white tags can be duplicated on a normal copier. Since it is possible for a Byte Tag to be duplicated without consent, never allow a tagged object to be the tool for unlocking a secure path in your application.
- Since Byte Tags are optical in nature they must touch the Surface screen in order to be seen. Hence, unprotected printed tags repeatedly touching and moving across the Surface screen may wear after a while and become less reliable.
- Similar to Surface responding to a user's touch, Surface must also respond to contact with a tagged item. Users require feedback to know that the system acknowledged and recognized the tagged object. Obviously, the type of appropriate feedback will depend on context. If a tagged object is to remain on the screen and users will manipulate other virtual screen elements to interact with the object, then the object will need its own persistent virtual user interface (e.g., a light ring around the object); the TagVisualizer (see [TagVisualizer](#)) control will be of great help.

Device Interaction

Tagged devices, like cell phones or cameras, are specialized cases of tagged objects. Tagged devices give applications an opportunity to exchange information with them via Wi-Fi or Bluetooth®. This is perhaps one of the greatest opportunities for an application to truly create a sense of magic and wonder, as long as any setup is minimal (if needed) and interactions are non-technical.



Figure 4.4: Example of a Byte Tagged device capable of wireless transferring data to and from Surface.

This particular capability is best achieved with a design strategy using the venue's own pre-configured devices (e.g., a set of Wi-Fi cameras owned by a hotel, which are already configured and tagged to work with Surface units in the venue). Alternatively, generic interactions with a user's own device does not involve tags, but rather the use of email or SMS text messages to his/her mobile phone.

Design Considerations

- Consider the physical objects that "naturally" exist in or near the target environment. For example, a Surface unit in a bar might be surrounded by cocktail glasses, stirrers, napkins, coasters, bowls of snacks, and candles. Are there ways to incorporate these objects?
- Microsoft Surface will be used as a real table and as such there may come a time when users need a respite from active participation. Hence, there needs to be a place or application to which users can navigate and know that they can use Surface as a table and not be bothered by unwanted prompts, sounds, or reactions and responses to every touch and contact. Fortunately, the default Water Attract application can be such a place. There are still signs of life, but nothing that would detract from people more focused on conversations and activities above the surface.
- If tagged objects are required in the design, consider the limitations of the placement of the tag on the object and the gestures required to perform an action. If an object is placed on the Surface screen incorrectly, then the system may not respond accurately, and the performance of the experience will feel broken.
- Physical objects should interact naturally with the application. Mimic and augment natural physical behaviors through the gestures that a user performs with a physical object and through the way that the Surface experience reacts to the physical object.
- Consider making the content in the Surface experience portable. Give users the ability to create content that can travel with them on a device, such as a mobile phone or Bluetooth®-enabled MP3 player. For example, allow a user to "flick" photos onto his or her Microsoft Zune® digital

media player or upload maps to his or her smartphone.

- Consider extending the Surface experience beyond devices and into a larger ecosystem. For example, in a retail environment, consider integrating the experience into the online experience for the retail shop.

4.2.5 End User Considerations

Designing for everyone is designing for no one. It is very important to have a target market and audience defined and understood. While we cannot comment specifically on your end users, it may be useful to at least consider the following distinctions between commercial and enterprise end users:

	Commercial	Enterprise
End user	General public	Knowledge workers
Training	None is expected or desired	Some is expected with the hopes of long term efficiency
Setup	None or very minimal is expected	Some is expected with the hopes of long term efficiency
Usage	Short-term	Long-term
Frequency	In short bursts only while at venue	Daily and potentially many times each day
Attraction	Internal curiosity and perceived benefits	External pressure (e.g. required work tool)
Priorities	Mixed (entertainment and accomplishment)	Productivity, efficiency, and accomplishment
Requirements	Simple, engaging, fun, empowering	Powerful, reliable, integrated

This information above is a generalization to illustrate a point. Hopefully, in calling out such differences, it is quickly apparent how some aspects of a design strategy would differ from one user group to the other. For example, given the short term system usage of commercial users, it is important to offer zero startup costs, as no user will want to invest any significant time or effort to learn or setup something they neither own nor will spend a great deal of time with. Hence, some features and capabilities may have to be exchanged in favor of simplicity. Whereas, the long term system usage of enterprise users suggests that they expect some cost in learning and setup, but to them it is worth the long haul benefits in productivity and efficiency. Now, imagine the impact on design strategy and planning with access to detailed knowledge of a target audience (especially within enterprise where there is a great richness of detail within any given specific work force).

Constantly ask yourself who, what, where, when, why and how? The last section (see [Methods and Tools](#)) of this document lists a great number of methods and tools to help in researching and understanding users.

4.2.6 Business Considerations

Choosing to create a Surface application is not always easy. While there is a defined benefit for the end user of ease-of-use and intuitiveness of the interface, corporate developers and designers may perceive the development process and lack of pre-canned solutions as a blocking issue, or one that requires investment beyond what they are willing to commit to. As such, the cost of and return on investment in designing and developing applications is a key consideration for software companies. Early adopters of new technology are traditionally willing to absorb the cost of developing, cutting edge products in exchange for being “first to market.” The high quality of Surface experiences requires a great deal of iteration, customer feedback, and rapid prototyping early in the process. In other words, the upfront investment to deliver Surface experiences can be significantly higher than GUI applications. However, Surface experiences can deliver a significant return on investment through reduction in cost-of-business operations (e.g., tasks that would otherwise require human interaction can be handled through a compelling software experience), and through increased brand loyalty. Benefits such as ease-of-use and higher user success rates can be hard to quantify, while development costs are easy to measure against a company’s bottom line.

Companies typically balance the cost of development (including the cost of hardware and the Software Development Kit [SDK]) against the benefits offered by the product, and compare that cost-to-benefits ratio with the cost and benefits of what they may consider to be “good enough” substitutes. Windows PCs currently enable touch experiences (every major OEM is developing a line of touch-enabled laptops, and many feature all-in-one solutions as well), and a significant investment is being made in the Windows 7 operating system to enable developers to build touch-enabled applications. While Surface technology enables the development of more compelling experiences than the PC platform (largely due to the integration of hardware components, such as the vision system), Surface design and development teams need to deliver Surface solutions where the balance of the cost/benefit equation weighs in favor of the Surface platform. To this end, full utilization of all key Surface attributes, as embodied in our UX Design Principles, is essential.

4.2.7 Brand Considerations

Great Surface experiences can elevate a company’s brand through transference. A company benefits through association, as consumers transfer the resulting emotions of an enjoyable Surface experience to the company or brand that brought it to them.

How should brands be manifested?

- Microsoft Surface provides a window into branded experiences. For example, in a retail environment the brand may extend beyond the merchandise and store presence to offer customers a glimpse of experiences in which their goods are used, such as river rafting trips at REI stores. In this way, the REI brand as an outfitter of outdoor gear and clothing, as well as adventure experiences, can be effectively demonstrated in stores and made more tangible to customers.
- Microsoft Surface extends other brand initiatives and enables customers to better understand what the brand stands for, which may be difficult to communicate in other ways.
- Microsoft Surface experiences become the expression of the brand in the same way that the Apple® iPod® wheel has become the “trademark” for the iPod itself. In this respect, the design becomes the brand language. The design of the Surface experience can become an expression of the brand.

- Similar to traditional approaches to brand development, it is important to understand the values you want your brand to represent. Microsoft Surface experiences can then facilitate those values through the experience.
- The user's total experience will affect his/her impression of your brand. Therefore, in addition to your great ideas and styles brought to life on Surface, it is important to get the fundamental Surface behaviors right and performant. If the system does not perform basic gestures and manipulations well (e.g., moving items, zooming in and out, rotating, flicking, etc.) it will dampen the overall experience and their impression of your brand.

Some additional brand related information:

- Microsoft Surface Co-Branding Guidelines (ask your Surface account representative)
- [Microsoft Surface Brand Identity Guidelines](#)³
- This document's [Visual Design section](#)

4.2.8 Microsoft Surface Software Environment

The Surface software environment consists of all of the general components surrounding Surface applications that complete the overall end user experience. These include attracting users to interact with Surface, navigating, going in and out of applications, and ending sessions. The following diagram shows the general structure and flow of the Surface habitat:

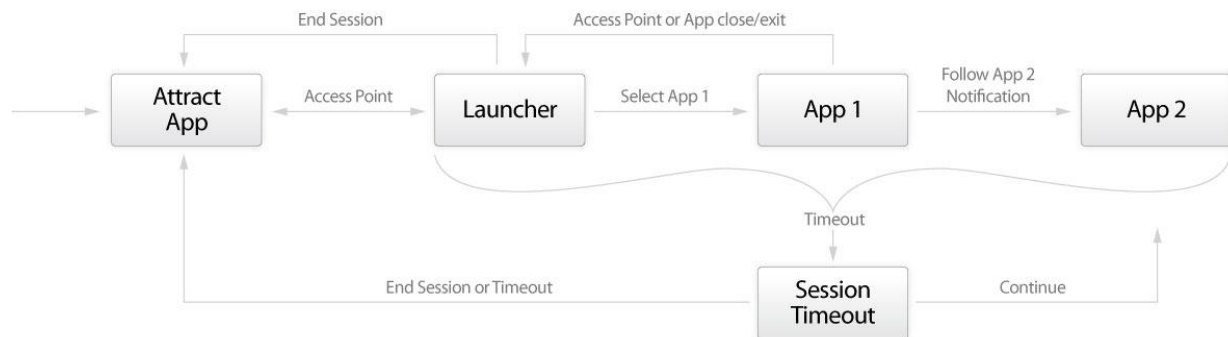


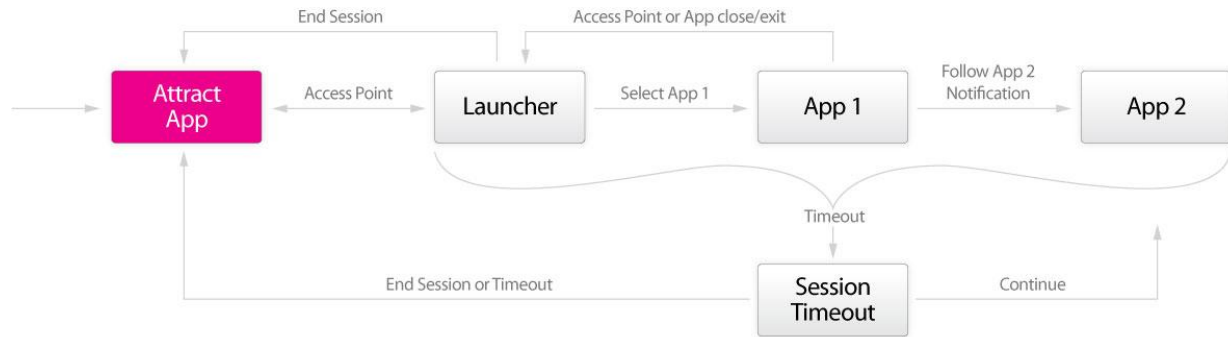
Figure 4.5: The components and flow of the Surface software environment.

The following sections discuss each component of this system to further explain the Surface environment. A different section (see [The Microsoft Surface User Interface Components](#)) provides design considerations and guidelines for customizable components of this software environment.

3

<https://brandtools.partners.extranet.microsoft.com/Corporate/Guidelines/Product+guidelines/Microsoft+Surface+identity+guidelines.htm>

Attract Application



When Surface is not in use and is in its starting or neutral state, the Attract application is running. As the name implies, the goal is to attract users to Surface and then compel them to touch and interact. Hence, the Attract application enables users to approach it and get an immediate response by touching and interacting with it naturally. The default Attract application (called Water) is a water simulation, which forms ripples when the Surface screen is touched. This is a terrific example of an experience that embodies the design philosophy. The interaction is natural and intuitive, there is an immediate response to every touch, it is immersive, and the experience mimics the natural world.

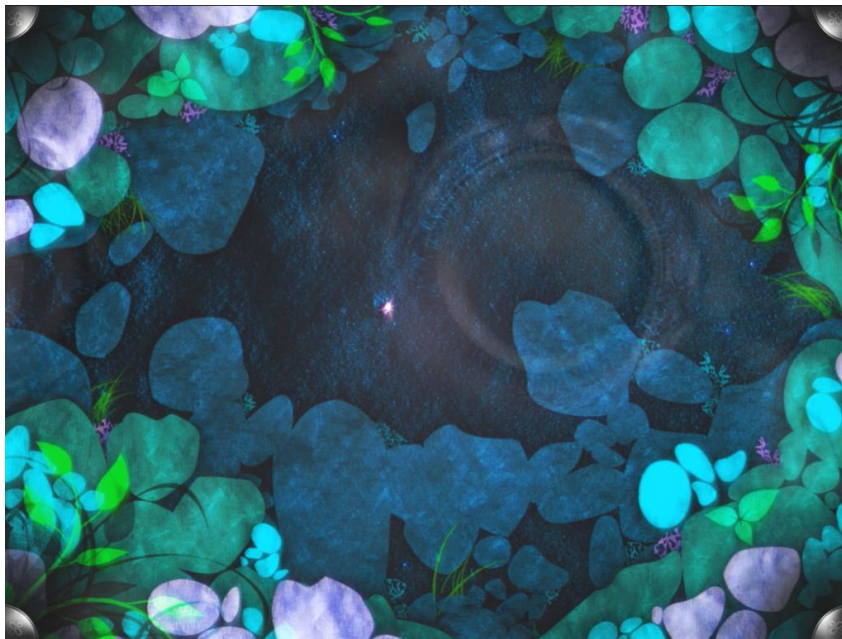


Figure 4.6: Default Surface Attract application is a water simulation.

There are several ways for users to navigate to the Attract application:

- When Surface powers on, the Attract application automatically starts
- When a user activates an Access Point on the Launcher screen (provided no other applications are running), the Attract application will return
- When the system resets to its neutral starting point:
 - When a user chooses to end a session from the Launcher

- When a users chooses to start a “New Experience” from the Session Timeout screen
- When a Surface session times out

There is only one way for a user to navigate out of the Water Attract application: he or she must touch or activate one of the four Access Points located in each corner.

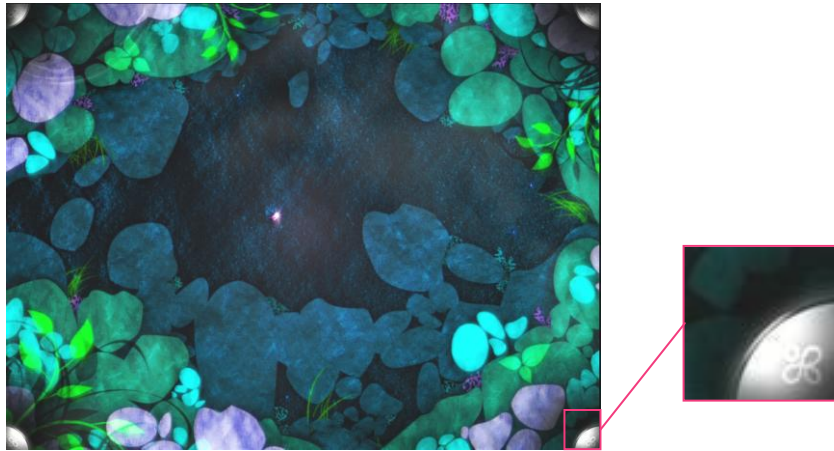
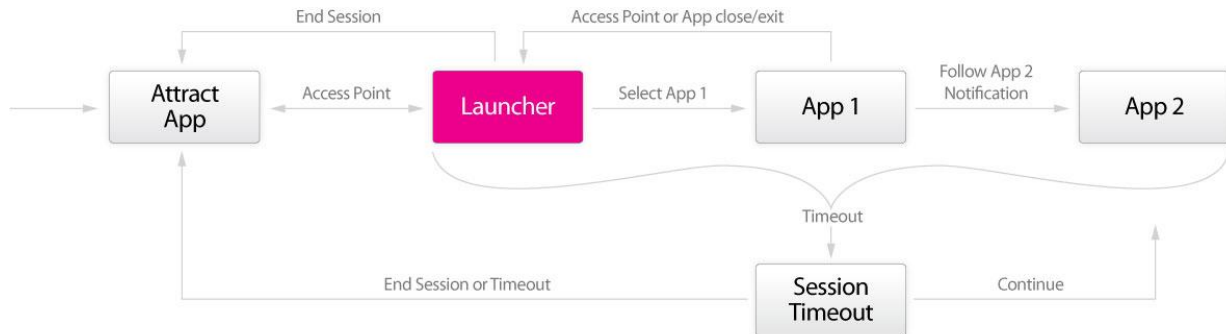


Figure 4.7: Attract application with Access Point buttons located in each screen corner.

Launcher



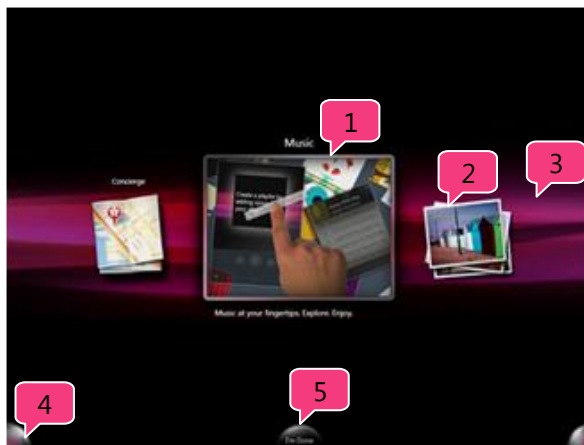
Launcher enables users to find and launch applications quickly. It was designed to handle scenarios involving two to fifteen applications. Although it could easily handle more, users may not want to scroll between twenty or more applications.



Figure 4.8: Launcher is used to start an application.

Launcher includes:

- Ribbon: stylized strip in the middle of the screen on which the application icons and the preview pane reside.
- Access Points: Surface-branded buttons used to navigate users between applications and the Launcher.
- Icons: large static image representation of an application
- Preview pane: a framed space within which images or video can be displayed
- End session button: only available on this Launcher screen, this button is used to close all running applications and return the system to its neutral starting point.



- 1 Preview pane
- 2 Application icon
- 3 Ribbon
- 4 Access Point
- 5 End Session Button

Figure 4.9: Launcher screen and all of its various components.

To see all the available applications, flick or slide the ribbon to the left or to the right:

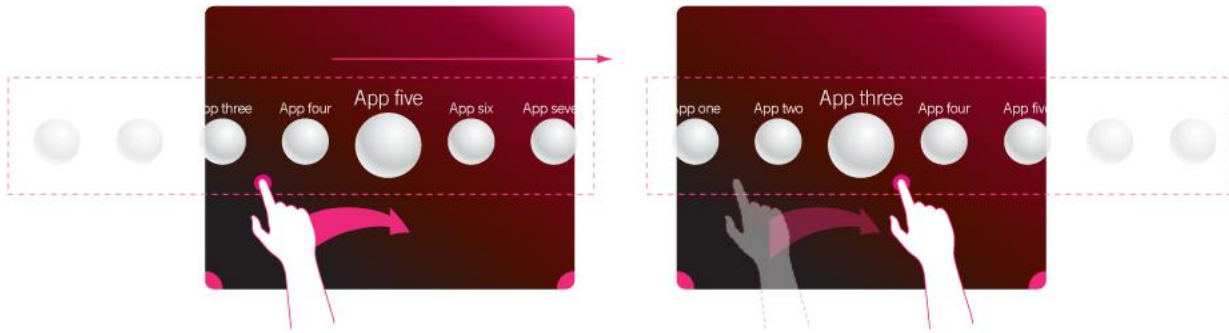


Figure 4.10: Example of the Launcher being scrolled from left to right.

Application icons that are in motion and passing over the center area will enlarge temporarily. An application that comes to rest at the center of the application ribbon will be displayed as a preview pane (see below). Also, touching an icon that is off to either side of center will cause the application ribbon to auto scroll the selected icon to the center and display the application as a preview pane.

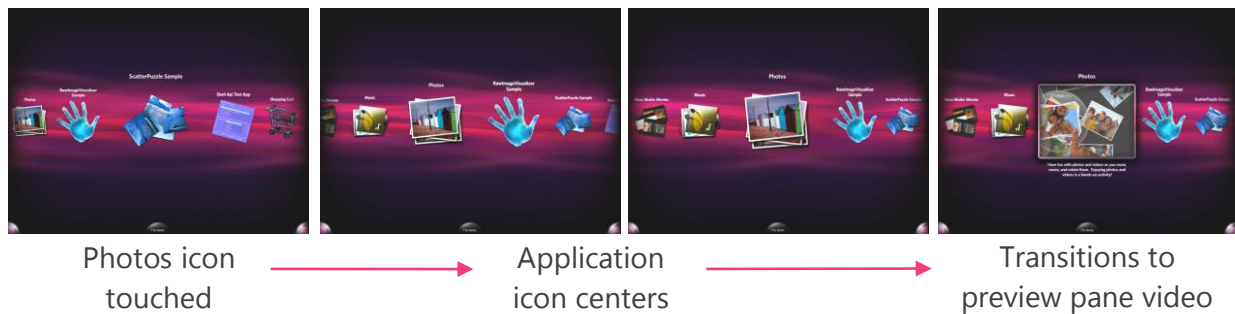


Figure 4.11: Application moving to the center of the Launcher and transitioning into a preview pane.

If there are fewer than six applications in the ribbon, then scrolling through the ribbon will not be necessary. Users can simply touch the application icon of their choice and it will convert into a preview pane.

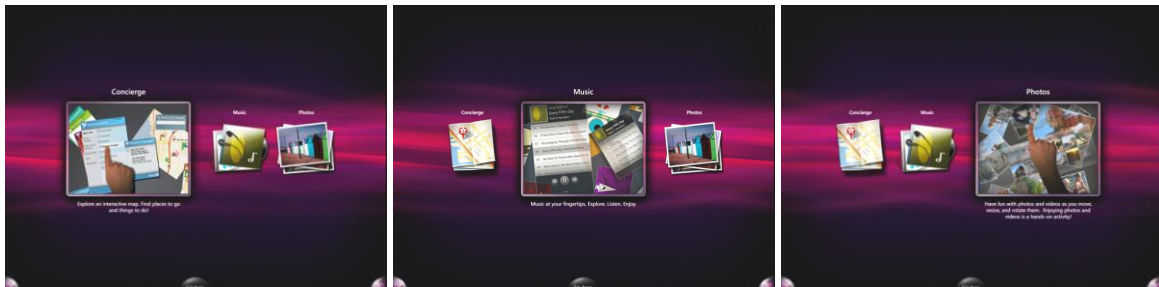


Figure 4.12: Touching each application icon turns it into a preview pane without moving it to the center of Launcher.

At the start of a new session and when Launcher is displayed for the first time, if users do not interact with

Launcher then after a short period of inactivity Launcher will go into “demo mode” and automatically scroll from one application to the next, showing each application’s preview pane video⁴.

The purpose of the preview pane is to provide additional information about the application beyond just a title and icon. Thus, a short video with audio can be played within the preview pane and additional descriptive text can be displayed below. The goal of the preview pane is to help users better understand what the application is and aid in their decision making. Additionally, providing a preview of various capabilities serves as a primer and a quick learning opportunity for users.

To open an application that is present in the ribbon, simply touch it when it is displayed as a preview pane. The preview pane will transition into an application loading screen, and the application will then open and fill the entire screen to invite users to immerse themselves in its “world”.

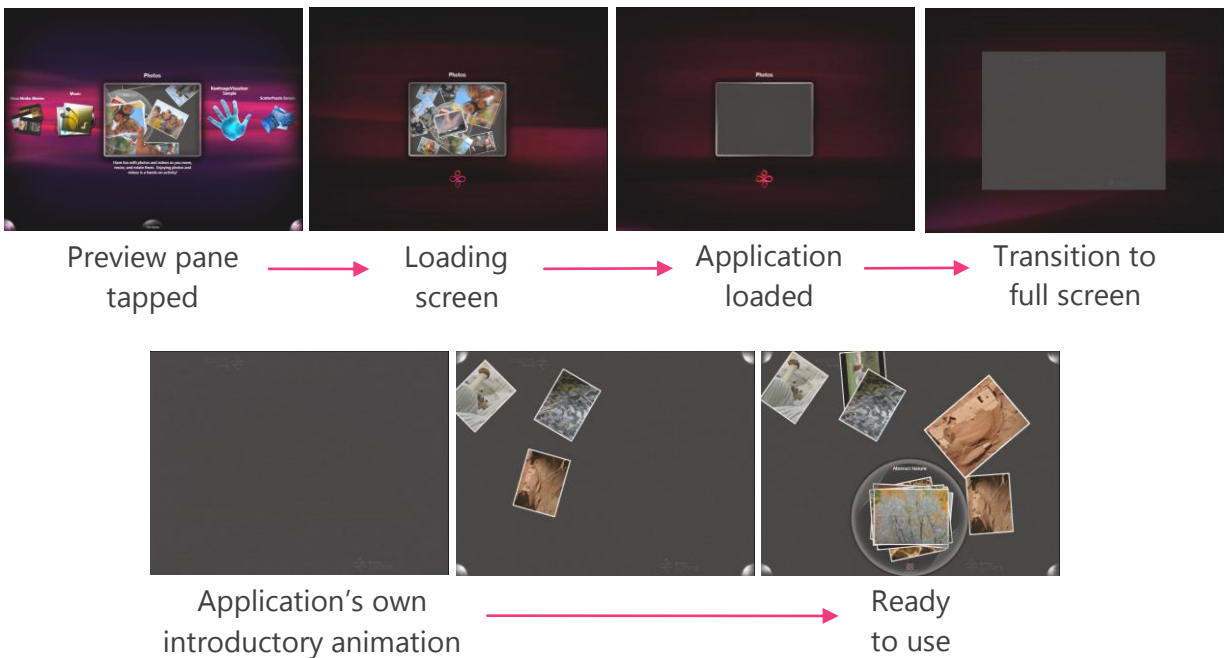


Figure4.13: Screen sequence illustrating the launch of an application.

When a user leaves an application running and returns to Launcher by pressing an Access Point, there is a seamless transition in which the application shrinks back into the Launcher’s preview pane. The application’s preview pane video is replaced with a “live” shot of the application:

⁴ While in “demo mode” Launcher will keep each application in the preview pane for a fixed time before automatically moving on to the next application. Thus, long preview videos may get cut off.

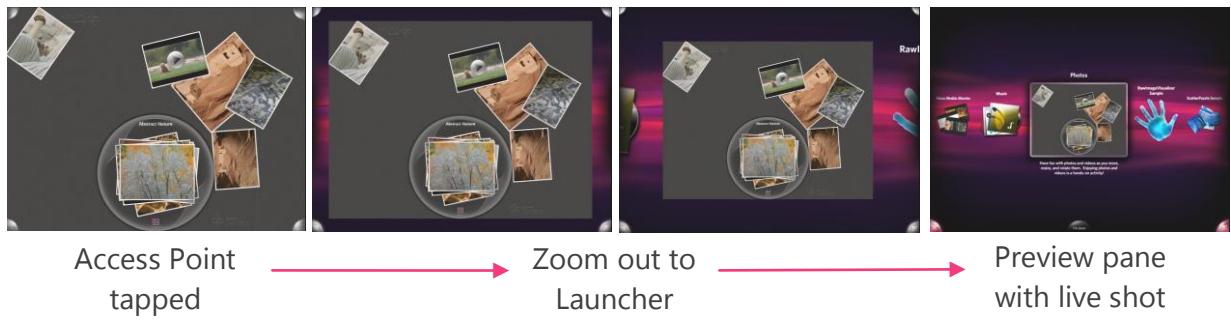


Figure 4.14: Back out of an application to Launcher and see a live screen shot in the preview pane.

Touching that “live” preview pane triggers a seamless reverse transition in which the preview pane grows to become the full-screen version of the application again:



Figure 4.15: The components and flow of the Attract application.

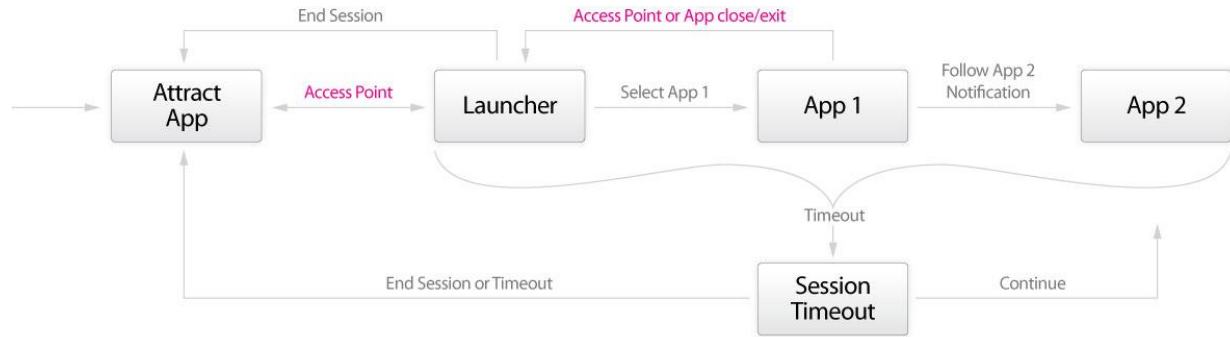
The seamless transitions helps users feel like they are zooming into and out of experiences.

When users are completely finished and ready to get up and walk away, they can end their entire session by activating a single button. The end session button will close all currently running applications and reset the system back to its neutral starting point, the Attract application. Given the potentially destructive nature of this button, a small safe guard was built in to avoid accidental activation: a two-step activation is required.



Figure 4.16: Ending a session is a quick two-step process.

Access Points



Access Points are key navigational buttons that are persistent throughout the entire Surface experience. Launcher features only two Access Points. They are always located below the application ribbon, whichever way Launcher is oriented.



Figure 4.17: Launcher has Access Points in only two corners of the screen.

In every application, on the other hand, there are four Access Points, one in each screen corner. They overlay every application so that users can return to Launcher at any time. The Access Points are semi-transparent and located in screen corners so as not to compete with applications for attention or valued screen real estate.



Figure4.18: Access Points are superimposed in all four corners of an application's screen.

Touching either of the two Access Points in the Launcher starts a smooth transition that opens the most recently viewed application, of all the currently running applications.



Figure4.19: Seamlessly zooming into an application from its live preview pane to a full screen experience.

Likewise, if the Attract application were the last viewed application, then touching an Access Point from within the Launcher screen will return users to the Attract application.

Touching any of the four Access Points in an application returns users to Launcher. Touching the Access Point closest to you also changes the orientation of Launcher so that it faces you.



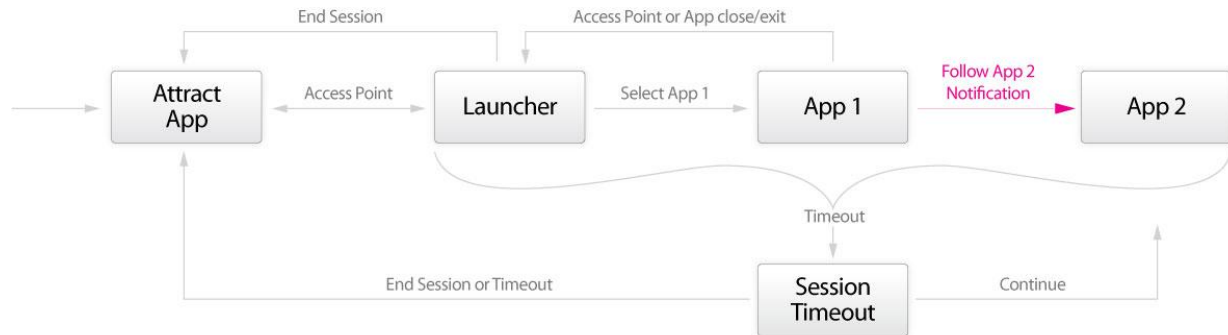
Figure 4.20: Launcher re-ori-ents based on activated Access Point within an application.

Note, that while the orientation of Launcher is dependent on the activated Access Point, the orientation of the application itself may or may not change; each application decides what to do for itself. For example, a game might choose to keep its original orientation to ensure everything continues to align and face players appropriately.

Finally, since activating an Access Point from within an application will navigate users back to Launcher, each application must decide whether to keep running, pause, or close itself. Here are some ideas to consider:

- A music application, for example, may continue running so users can enjoy listening to music as they engage in other applications. In such cases, designers should be mindful that users may have to navigate back to the application to adjust settings (e.g., change volume, skip to next track).
- A movie or game application, for example, may choose to pause. In such cases, the application should be careful to consider timing when it resumes. As users navigate back into a paused application, the zoom-in transition animation takes a brief second, but during that time, users cannot interact with the application. An application should be careful to only fully resume after the zoom-in transition is done (about $\frac{1}{2}$ a second).
- Closing an application is not recommended. It is important for users to learn and expect Access Points to behave as navigation elements, not as “exits.” Users should navigate away from an application knowing that when they return to it, they will find it as they left it. Additionally, users on occasion will accidentally activate an Access Point and if the application were to close, then users would lose their work and have to sit through reloading the application if they start it again. If it is important for an application to close or reset, then it should provide that functionality as an option for users to explicitly choose and activate from within the application itself. The exception is for applications that are stateless, such that users did not expend any effort or time to get the application to a specific point worth saving (e.g., Water Attract application).
 - Under special circumstances it is possible for Launcher to hide all Access Points: The first time an application is successfully launched from the Launcher screen, the system will close the Attract application. If that first application was subsequently closed (either voluntarily or due to a failure), then there are no more applications running and so Access Points are not needed as there is nothing to return to.

Notifications and Errors



Notifications are Surface’s way of enabling applications to send messages to users. A Notification consists of a preview pane containing a live screen shot of the application sending the message, a dismiss button, a message title, and a message body:

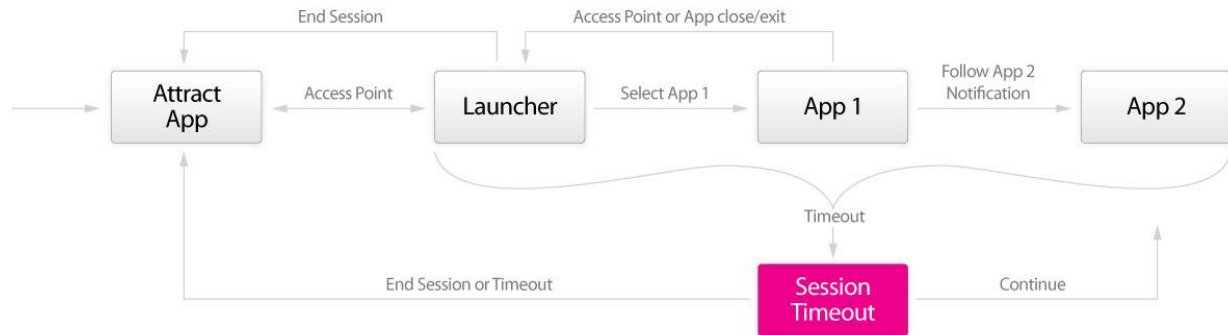


Figure 4.21: Different components of a notification.

Generally, these messages come from the system or from other background applications or processes. Applications can choose what type of messages or information to send via Notifications. The system will use Notifications to inform users of errors (e.g., an application failure).

Notifications show only on one side of the screen—the side that Launcher was facing the last time it was displayed. Notifications animate onto the Surface screen by sliding out into view along a long edge of the screen. If ignored, a Notification will time out and slide back out of view. Alternatively, users can manually dismiss a Notification by activating its top, centered button, or by dragging / flicking it back off the screen. Additionally, users can touch the Notification’s preview pane as a way to quickly navigate to the application responsible for sending the message. Similar to the behavior of Launcher, touching the Notification’s preview pane triggers it to smoothly transition from a small live screen shot into the full-screen application.

Session Timeout



With commercial venues in mind, the Surface software environment includes a session timeout feature to account for users who have finished or abandoned their session and walked away without explicitly ending their session.

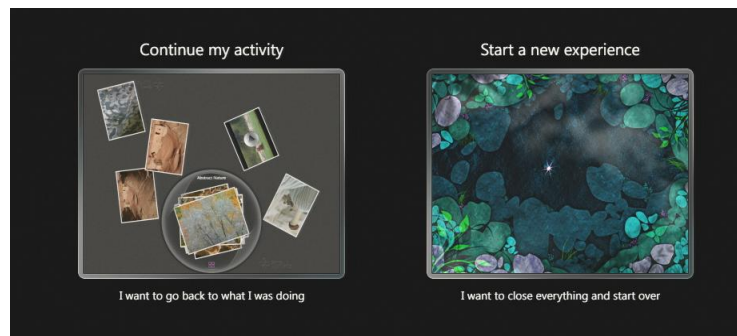


Figure 4.22: Sample session timeout screen.

After a specific period of inactivity (timing is customizable) the system does a zoom-out transition to a Session Timeout screen, offering users a choice:

- "Continue my activity"—Surface will simply zoom back into the current session.
- "Start a new experience"—Surface will close everything and restart at the Attract application.

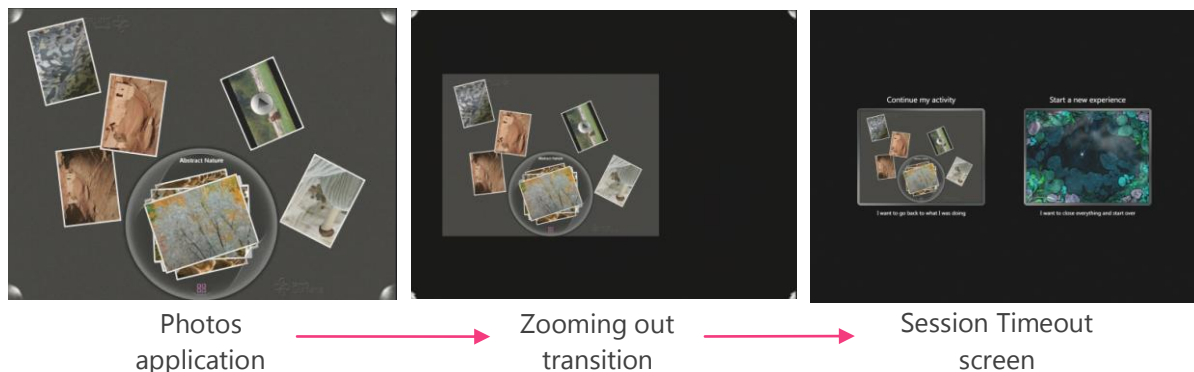
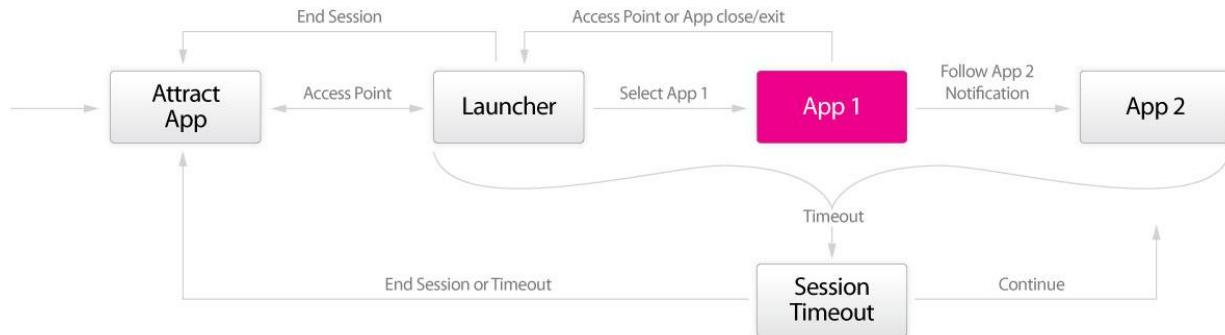


Figure 4.23: Inactivity eventually leads to a Session Timeout screen.

Note that the system will broadcast its intent to time out a session, and at that moment an application can respond and decline the timeout, before the user is interrupted. Applications need such control in order to make intelligent decisions. For example, an application would not be intelligent or contextually "aware"

if it periodically interrupted users with a timeout screen while playing a long video not requiring any user interaction.

On-Screen Keyboard



The on-screen keyboard is also a component of the Surface software environment and it appears only within applications. The text input control is currently an on-screen keyboard that is optimized for touch and appears whenever a text box type control receives "focus":



Figure 4.24: Sample screen shot showing a text box and the on-screen keyboard.

Microsoft Surface has two related controls, `SurfaceTextBox` and `SurfacePasswordBox`. They automatically handle showing, placing, and configuring the on-screen keyboard when the text box controls get focus. The `SurfaceTextBox` turns on the keyboard's key press visual feedback so the user can see what key he or she is pressing, and the `SurfacePasswordBox` turns off the visual feedback (for obvious security and privacy reasons). It is possible to override this behavior, but not recommended.

The on-screen keyboard is semi-transparent and appears as a full alphanumeric keyboard or as a numeric keypad:



Figure 4.25: Sample screen shot of the on-screen numeric keypad.

Users can move and rotate the keyboard by touching and dragging from the top “handle” area only. This is necessary to avoid having the keyboard unintentionally move as users type.

When interacting with the on-screen keyboard, multiple inputs are accepted. Each input is treated as a separate touch, and multiple keys can be depressed simultaneously, similar to a physical keyboard.



Figure 4.26: On-screen keyboard close up showing two keys pressed simultaneously.

Since Surface is a multi-touch system, the on-screen keyboard does not automatically disappear when users interact with other objects on the screen. Users can dismiss the keyboard by touching the keyboard’s own “dismiss” button (centered at the top) or by finishing the input task (e.g., activating an application’s “Search” button after inputting search text). Furthermore, only one on-screen keyboard will appear at a time, and it will send its input to whichever text box has focus.

Out of Order Screen

When the Surface unit is out of order, then the following is displayed full-screen:



Figure 4.27: Sample out-of-order screen.

Additionally, this screen might appear when an administrator remotely logs onto the unit for maintenance. This screen indicates that the unit is not currently working.

If a user were interacting with the system at the time it experiences a major failure, then the Surface unit first temporarily displays an error screen to indicate that an error occurred, and then the unit displays the Out of Order screen until the unit is fixed.

5.0 Interaction Design



Microsoft Surface proposes a new relationship between humans and technology, one that unfolds intuitively through natural human input. This relationship is given space and form through the practice of, and attention to, interaction design. This section will cover interaction considerations relating to Surface design principles, an overview of gestures, and a high-level interaction design approach. The ultimate goal of this section is to provide conceptual and practical guidelines for ensuring that Surface experiences take full advantage of the promise of a natural user experience.

5.1 Approach

Interaction: A Focus on Users

Interaction design facilitates the relationship between users and a system, elegantly marrying user behaviors with product responses in context and over time. Thus, interaction design begins with a focus on users: who they are, what they want, and which computing experiences frustrate or delight them, all in the interest of gaining insight into their goals, needs, tasks, dreams, and relationships. Relationships are of particular importance, as Surface experiences emphasize social interaction. Research and design tools (see [Methods and Tools](#)) provide the framework for developing a solid foundation for the heart of a Surface experience: the design concept.

Familiar Metaphors

Identifying users' underlying mental models, or their understanding and expectations of a given experience, is particularly important in upfront exploration. These models carry inherent metaphors, processes, and expectations, and can range from subtle to explicit. They form the basis for a successful Surface experience, and enable both quick learning and the opportunity to delightfully expand beyond real world experiences. For example, a Surface user in a music store may have a particular genre in mind (organizational model), search for a specific album (physical metaphor), ask an employee for a recommendation (relationship) and ultimately purchase some CD's (process). Placing a physical album on Surface could result in related virtual albums to spill onto the screen. The user could open and sample the music on the store's sound system by touching individual songs. At the same time, the music video could replace the album's cover art. These interactions leverage familiar metaphors, yet go beyond them to create magical interactions to create a surreal experience. The key is to avoid overly mechanical, abstract, and confusing metaphors.

Interaction Prototyping

After achieving a deep understanding of people in a social context, and forming design concepts rooted in familiar metaphors that provide magical possibilities, prototyping can then begin in earnest. Testing concepts early and often in low-fidelity media (marker, paper, sketches, storyboards, loose wireframes, semi-linear touch-throughs, etc.) allows the team to quickly discard unsuccessful directions, rapidly iterate to improve promising concepts, and even involve users in the creative process to discover new directions. For Surface, low-fidelity, tactile methods are particularly relevant as they allow users to work with touch.

Interaction Modeling

Once a design concept has been thoroughly prototyped, validated, and selected, then it is time to specify

all the elements of the interaction design. Interaction models outline the underlying structure and process in a design. Task/flow diagrams articulate how an experience progresses over time. Wireframes provide detailed snapshots of significant sections or moments in the experience. Object state/transition diagrams account for every element, their relationships, and their states. These final interaction design specifications are the underlying blueprint for the Surface experience, and are combined with visual, motion, and audio design in order to bring the final experience to life.

5.2 Guidelines

The following pages outline interaction design guidelines and considerations, organized by the core Surface design principles (see [Vision and Philosophy](#)).

5.2.1 Seamless

The application is comfortable to interact with.

Users interact with the application according to what they already know about the real world. As a result, features and content are easily accessible and clearly understood. It is always clear how to begin, end, and make changes. For example, a group of photos arranged in an organic yet organized stack communicate that these can be moved around the screen together, or flipped through as in the real world. When evaluating the interaction design, ask:

- Are controls for starting, ending, and major state changes always visible?
- Does the application bridge comfortably to what users already know about how to interact with objects in the real world?

The physical and virtual are integrated.

Erase the line between the physical and virtual worlds. The Surface experience must respond continuously to fingers and physical objects that are placed on it, create an environment that suspends disbelief, and immerse the user in a better-than-life experience. For example, playing with and putting physical objects on the Attract application provides both a familiar and magical experience. When evaluating the interaction design, ask:

- Are familiar metaphors with magical capabilities being utilized? Are web or PC interaction models being used where new models could improve the experience?
- Does the application's behavior and physics model evoke the real world, with a sense of being organic, nuanced, in-depth, and providing immediate feedback?
- Are there appropriate object behaviors that encourage rich multi-touch?

Physical objects enrich the experience.

Surface experiences are not limited to the screen. Physical objects can be recognized, and the system can respond to them according to their placement or movement. Consider seamless interactions between physical objects and the application, as well as new gestures that blur the line between the physical and virtual. For example, placing a glass or coaster on top of the Surface screen could cause virtual items to scoot out of the way. When evaluating the interaction design, ask:

- Are physical objects used in the application?
- Are the application's responses to physical objects appropriate to both the object's properties

and user's expectations for the experience?

- Can the application still be reasonably used when objects are absent or misplaced?

Transitions are seamless.

Transitions contribute to both pleasure and understanding. This is true of significant transitions, such as getting started, changing activities or modes, and closure at the end, as well as minor transitions, such as showing that a content item has been selected. For example, moving from an application icon in Launcher to the application is a completely smooth transition, without additional splash screens or loading bars. Meaningless transitions, such as changes in the appearance of content that do not correspond to a change in state, should be avoided. When evaluating the interaction design, ask:

- Do transition animations communicate state changes, relationship changes, or contribute to a consistent overall personality of the application?
- Do objects appear and disappear out of nowhere, or do transition animations maintain continuity, and help users stay oriented?
- Do the motions and behaviors of objects simulate motion in nature, demonstrating mass, acceleration, friction, gravity, and viscosity?

5.2.2 Social

The experience comes alive with several users.

Microsoft Surface experiences are not only about the interaction between user and interface. They are also about the interactions between the users. Multi-user content and controls can occur simultaneously. Give users toys and tools for interacting with the application together. For multi-friend scenarios, avoid imposing rules; allow users to work out social factors on their own. However, do not expect the same for multi-stranger scenarios. For example, in a bowling game you can start by jumping right in and playing, or set up the players first. If you play first, it will ask you who you are at the end of the game and anytime during play the game can be reset. When evaluating the interaction design, ask:

- Does the application allow content and controls for more than one user at the same time?
- Are the gesture set, layout, and content presentation optimized for many fingers and many hands?
- Can a single user still reasonably utilize the application if needed?

The interaction is 360°.

Content and interface elements clearly appear, orient, react, and disappear for all users. The application treats everyone as equally worthy of its attention, while still providing a single collaborative experience. For example, when content first appears in the Music application, albums are scattered across the screen in different orientations, allowing for interaction on all sides. When evaluating the interaction design, ask:

- When new content and controls appear, do their motion, orientation, and size make them feel usable by anyone within reach? When this is not the case, is it only because of tradeoffs for the system to be aware and responsive?
- Is the application equally easy to use from any side and from more than one side at the same time? Does user orientation cause content or the interface to adapt when appropriate?
- Does the application enable a social, collaborative scenario, or is the screen being carved up into

many single-user regions?

It is always clear when and how another person could join.

Through orientation of content, provision of controls, and overall behavior, the application never acts like the screen is the exclusive territory of a single user. Should an additional user find it socially acceptable to join, it should be clear that the application would welcome that. For example, a single-player game can transform into a multi-person experience as soon as touches are detected in new locations by using contact orientation data. When evaluating the interaction design, ask:

- Does the application always provide a way for another user to join?
- Does it welcome new users, either explicitly or implicitly?
- Does the experience become richer with more users?

5.2.3 Spatial

There is a feeling of depth and immersion.

Graphical User Interface (GUI) interaction models are flat. Surface interaction models go beyond a simple plane to provide depth and encourage immersion. This does not mean that 3D environments should always be used. Sometimes they are disorienting and overly complex. It simply means that the application behaviors, transitions, and navigation take the z-axis into consideration. For examples, content in the Photos application are inherently resting on a flat canvas, yet rise to the surface when touched to give the feeling of depth and realism. When evaluating the interaction design, ask:

- Are object behaviors and movements constrained to a rigid, single plane?
- Do the overall environment and elements support transitions and movements that allude to the z-axis at different states (at rest, when touched, when transforming)?
- Do multiple objects interact with each other in believable ways, according to a subtle and rich physics model?

Screen space is used appropriately.

Depending on an application's scenarios and context, the viewable space may be more or less constrained. In some cases the canvas will be fixed, with a limited content presentation. In others it will be flexible, allowing for zooming in and out. Leverage spatial memory in situations where the canvas is larger than what is in view within the screen. In either case, backgrounds, objects, and controls take the z-axis into account for their behaviors and movements. For example, the Concierge application allows for an infinite canvas when navigating the full screen map, yet content cards and controls stay within the boundaries of the screen. When evaluating the interaction design, ask:

- Do the application scenarios require a fixed or flexible canvas?
- Are the navigation capabilities communicated through object behaviors and the overall interaction model?
- Can the application become too cluttered or too sparse? How can the space interaction model ease extreme cases and ensure social scenarios are respected?

There are layers of discoverable detail.

From a distance, objects, content, and other elements stand out, and are clearly recognizable and iconic.

When viewed closer, more detail is revealed, such as additional information, subtle textures, or hints of reflected light. When users interact with interface elements, they reveal yet a finer level of detail through sound, visual feedback, and movement. For example, icons in Launcher transform into application preview movies when touched, then into the live application when touched again; all providing progressively more detail with deeper interaction. When evaluating the interaction design, ask:

- As users zoom in closer to objects, do the objects reveal unexpected visual or audible details?
- Do these details provide a pleasant surprise, new information, or both?
- Are changes in content communicated through transitions in the appearance of content objects?
- Do visual and audible changes (other than light and shadow effects) correspond to real changes in the underlying information structures?

5.2.4 Super-Realism

The experience is natural and better-than-real.

Super-realism is achieved through rich, detailed representations of familiar real-world behaviors, augmented with delightful capabilities. These two goals are deeply interconnected, as magical capabilities will be more believable if they emerge out of a real environment. For example, photos in Surface that are easily enlarged through gestures cannot be stretched in the real world. When evaluating the interaction design, ask:

- Does the experience feel organic, fluid, and smooth?
- Are objects reacting naturally in their physical behavior?
- Are real-world behaviors represented in every appropriate detail of physics, motion, reactions, collisions, and sound? Are they also augmented in intuitive and magical ways?

Microsoft Surface is always alive.

The application's interface is rarely at a complete standstill, always showing signs of life. Moving and changing elements are pleasantly surprising and unobtrusive. Find the Surface-appropriate equivalent to a person breathing or blinking, or clouds gracefully passing in a summer sky. These things are constantly present, but never distracting. As in the real world, every touch produces a response, however subtle. Fingers, hands, and objects should always be met with an immediate reaction, making the experience feel alive, and giving the user constant feedback. For example, the Water Attract application is in constant motion, always inviting users to touch and engage with the experience. When evaluating the interaction design, ask:

- Does the application always show signs of life, even without user interaction?
- Is it subtle enough to avoid annoyance and distraction?
- Does the application provide constant feedback with every touch?
- Are fingers, hands, and objects recognized and responded to appropriately?
- Does the experience allow playful or exploratory touches, as well as primary, expected gestures?

Users feel progressive delight.

Over time, interaction design for Surface should provide a framework that allows users to gradually encounter capabilities. Begin the experience with familiar metaphors and objects, and then expose

additional possibilities as the interaction unfolds. A careful attention to this balance will allow for progressive delight, and draw users deeper into the experience. For example, the Water Attract application begins with gentle ripples to entice users, then responds to every touch, and ultimately draws attention to the Access Points in the corners to allow for deeper engagement. When evaluating the interaction design, ask:

- Does the experience begin with a familiar environment and behaviors?
- Can users quickly discover delightful interactions?
- Does the experience provide continued delight and discovery over time, in minutes, hours, days, or months?

5.2.5 Contextual Environments

Microsoft Surface is always aware and responsive.

Microsoft Surface is always ready and does the right thing at the right time. The application feels alive. It reacts instantly to any touch, and seems to sense where the user is. There is always something else to touch, but nothing to break the flow of attention. The sights, sounds, and reactions give a sense of having direct, precise influence on something that has an energy of its own. For example, the Firefly game gives immediate feedback through glow effects with every touch, gestures are directed through the behavior and responses of the fireflies, and the physical metaphors of grass and glass jars create a believable, immersive environment. When evaluating the interaction design, ask:

- Is the response time for all actions perceived as instantaneous?
- Is the application's first response priority to provide user feedback?
- When designing games dependent on user reaction time, can system performance keep up?
- Do transitions begin with near-zero lag and is the pace consistent across the entire experience?
- Consider the technical performance between Bluetooth®-enabled devices. Is there any performance lag between the devices to break the continuity of the performance?

Subtle affordances invite exploration.

The majority of Surface experiences are highly exploratory and contextual to the environment. Experiences should invite users to interact as they are comfortable, rather than telling them what to do. Consider designing subtle affordances that invite users to discover through exploration. For example, the queue in the Music application gently pulses when music is playing, drawing attention to this important interaction space where playback and customization occurs. When evaluating the interaction design, ask:

- Do objects and content do what users think they should?
- Can users easily discover what to do and where to go through subtle but clear cues?
- Are subtle affordances provided on-demand and in context?

Microsoft Surface is intelligent but not presumptuous.

Anticipate responses to simplify and create efficiency, but keep the user in control. For example, in the context of interactions about a recent trip to Europe, anticipating users' wishes suggests automatically displaying their vacation photos. However, doing so automatically can be presumptuous, disruptive, or annoying when wrong. Instead, suggest the choice to display vacation photos, thus making it easy and quick for the user to do the next anticipated thing, but leaving him or her in full control. When evaluating

the interaction design, ask:

- Is there a chance a well-intended, anticipatory response can be wrong or undesirable?
- Is the application learning about users' intents with every touch, and gently guiding at every step?
- Are users in full control all the time?

The experience orients to its users.

If the application has a dominant orientation, direct the interface towards the user who made the touch. This applies to the overall screen, as well as individual interface elements such as menus. If the content can be oriented in any direction, use the direction of the first touch to influence its initial orientation. For example, the Concierge application presents text-based content towards the dominant side to provide for easy legibility, while the Photos application spreads out visual content in a more scattered arrangement. When evaluating the interaction design, ask:

- When new content or interface elements appear, does the application orient towards the user whose touch triggered its appearance?
- When multiple pieces of content or interface elements appear, does the application orient them to be inclusive of all users around the screen?
- Is the application using contact orientation information appropriately, displaying individual content items towards each touch, and adapting to the environment as needed?

5.2.6 Scaffolding

The experience is focused.

Microsoft Surface experiences trim features and focus on a few immensely rich options. Consider narrowing the number of choices to the bare essentials, and focus on making them the most rich and rewarding experience possible. Fewer choices and enhanced details can create a rewarding and pleasurable experience that makes users feel as if they are enjoying something of the utmost quality. Applications with a high number of complex options and features are not ideal candidates for a Surface experience. For example, an immersive slideshow could emerge from playing with photos, but a tag editor would likely be avoided. When evaluating the interaction design, ask:

- Is the application trying to achieve something too complex, or do too many things all at once?
- Are users always given simple, clear actions to take?
- Is the interaction rich enough to make simple scenarios engaging?

Learning is integral with doing.

The application contains levels of depth that smoothly ramp users from their first touch to full engagement with the application. If instructions are necessary, they are integrated with the natural flow of use, and do not steer attention away from content. For example, when a button triggers an event, a gesture that results in the same event could be included as a visual overlay. This educates the user how to accomplish the same result gesturally. When evaluating the design, ask:

- Do the first touches reveal the application's features and behaviors?
- Do any essential features require more than one touch to be discovered?
- Are instructions, if any, presented in the natural flow of interaction, with minimal distraction from

the main activity?

Progressive disclosure unfolds capabilities over time.

When leveraging the spatial capabilities of Surface experiences, progressive disclosure can be achieved by navigating through the environment or providing on-demand, deeper views of an object. For example, users can zoom into a photo to reveal deeper content and functionality. When evaluating the interaction design, ask:

- What interaction possibilities are essential at each moment? What is the most important content or capability to expose at a specific time?
- Is the application subtly but clearly hinting at deeper possibilities, but not taking the focus away from the content or primary focus?
- Can the application be navigated quickly and easily, while still respecting social considerations?

Users never feel like they've failed.

Users may try something that does not work, but the resulting feedback helps them learn, resolve problems, or encourages them in a fruitful direction. For example, bugs in the Firefly game simply become inactive when touched too harshly, subtly teaching the user that a gentle touch is needed. When evaluating the interaction design, ask:

- Does anything mislead users to mistakenly believe it is touchable or functional?
- Does feedback contribute to a better understanding of how to use the application?
- Do users always feel in control?

5.2.7 Performance Aesthetics

The aesthetics of Surface experiences are not just in the visual representation of the interface, but also the mechanics and behaviors of the objects and content with which users interact. Consider the detailed motion and physics of an element's movement and behavior.

The experience is playful and pleasurable.

Endow everything with the utmost quality and enjoyment. Microsoft Surface experiences focus on highly crafted representations of content, motion, and sound to help create fun, pleasurable experiences. The interaction design should not focus on mere task completion, but on the quality of the journey itself. For example, the transition from a stack view of photos into a grid view supports the task of browsing more content, but occurs with every detail of seamlessness: a smooth expansion of the container's shape, seamless resizing and movement of individual photos into the grid, and gentle motion on the content to indicate scrolling. When evaluating the interaction design, ask:

- Are behaviors purely mechanical, or are the mechanics endowed with subtle charm?
- Are users enjoying both the clear results of their actions and the actions themselves?
- Are users having fun in a social context?

Appeal to multiple senses.

Where possible and appropriate, the application appeals to more than one sense at a time. Visuals, sound, motion, and physical interactions serve to communicate, create mood, convey personality, direct users' attention, and enchant. For example, the Access Points are highlighted with light effects, audio cues and

collision ripples within the Water Attract application. When evaluating the interaction design, ask:

- Where possible and appropriate, does the application appeal to more than one sense at a time?
- In doing so, does it take a balanced approach so as not to overwhelm or seem unnatural?
- Do the various physical, visual, and auditory responses seamlessly integrate to create a single yet complex effect?

There is instant gratification.

All actions should have immediate and obvious consequences. The “busy” mouse pointer is an artifact of the past and should not exist in Surface. Good experiences do not make users wait for even a moment because it breaks their immersion, flow, and concentration. Make the performance of the experience flawless and understand the technical constraints. Performance is only as good as the slowest element in the experience. For example, the water Attract application was carefully optimized to achieve an excellent balance between visual quality and frame-rate performance. When evaluating the interaction design, ask:

- Are behaviors rich enough to be compelling, but simple enough to be feasible?
- Do users ever feel like they’re waiting for something to happen?
- Does the application feel slow and clunky?

5.2.8 Direct Manipulation

The nature of direct manipulation invites designers to develop an entirely new interaction vernacular that changes point-and-click to touch and gesture. Consider continuous gestures performed by fingers and physical objects that are optimized for direct manipulation.

The experience is fluid.

One striking difference between a standard computer interface and Surface is the seamless continuity that comes with direct interaction and high degrees of freedom. Far from brief, single-click, sequential actions, movements in Surface are fluid and dynamic. This is not only true with simple movement of objects, but also with other gestures that allow for transformations of content, such as opening, closing, flipping, sending, etc. For example, cards in the Concierge application smoothly flip and resize to present more information. When evaluating the interaction design, ask:

- Are objects designed to respond to rich interactions (i.e., touch, multi-touch, and gesture), or could they just as easily have been clicked by a mouse?
- Are transitions barely perceptible, providing smooth transformations from one state to the next?
- Is the environment optimized for touch in layout, feedback, metaphors, and behaviors?

The content is the interface.

Users directly move objects on the screen. Controls may reveal themselves from content, but non-content user interface is limited. Controls are lightweight and are relevant and necessary for the content. At all times the application preserves the illusion that users are interacting directly with the content itself. For example, the touch areas of the Stack control in the Photos application are at the top of the z-order when at rest, but are partially obscured when photos are being touched within the stack. When evaluating the interaction design, ask:

- Is the content itself the primary means of interacting with the application?

- Are controls only provided when necessary, in context, and associated with content?
- Are gestures and behaviors appropriate to the content in terms of weight, friction, size, and transformation?

Users have control.

As appropriate, distinguish between the touch of a finger, the brush of a sleeve, or the placement of a cup. When technical limitations create the chance of ambiguity, either require confirmation or make it easy to undo. For example, major operations such as ending a session or clearing all content should require confirmation. When evaluating the interaction design, ask:

- Does the application accurately distinguish between different possible types of contact, such as a finger, a hand, or an object?
- Does the application make it difficult for users to unintentionally cause large changes or transitions?
- Does it make it easy for users to correct or otherwise recover from unintentional actions?

Interaction is free-form.

Encourage the discovery of content through exploration. The more a user explores, the more content and functionality are revealed. The application offers a world of content and functionality that users can engage as they choose. The design prefers open exploration to scripted or sequenced interactions. There may be multiple ways to accomplish the same goal, many possible ways to arrange content, or a natural, open environment influenced only by user actions and its own internal rules. The growing popularity of semi-structured or fully free-form environments in the game industry is a good example of the types of experiences that would work well on Surface. When evaluating the interaction design, ask:

- Does the application avoid scripted or sequenced interactions where possible?
- Does the application provide an open, free-form environment for users to manipulate?

5.3 Gestures

Gestural interaction with Surface is a field with ongoing development. In this section, we first describe the types of gestures that are possible with the current platform, and then describe the gestures which are detected and made available.

Gestural interaction is the preferred user experience with Surface. The principle of direct-manipulation is particularly salient: gestural interaction should remove layers of abstraction separating a user from his or her tasks. Rotating an object, for example, is best achieved by grabbing it and turning it to the desired position. Gestural interaction allows this to happen naturally, without the need for buttons or text entry which might distract from the primary task.

5.3.1 Gesture Parameters

Point-Based Gestures

Gestures on Surface are limited by the sensing capabilities of the system. Each touch is sensed as a set of *contacts*. For each of these contacts, the system reports to the application its location (x and y coordinates,) orientation (absolute angle relative to Surface,) and size. The actual geometry of the area of contact is not reported. As a result, gestures in Surface are limited to changes in the positions of contacts

over time. These gestures are known as *point-based gestures* because subtle changes to the area of contact, such as changing the posture of the palm or fingers, cannot be sensed. As a result, gestures which rely on sensing differences in initial posture are not possible. For example, the system cannot reliably distinguish between a fist and palm contact:



Figure 5.1: Microsoft Surface reports touch as a set of contacts. Contacts do not include the geometry of the touch area, so subtle differences in posture, such as the difference between a palm and fist, cannot be reliably detected.

In both of these cases, the system reports only the contact position, orientation, and size; there is not enough data to reliably distinguish between these postures. To reliably differentiate initial postures, gestures must distinguish only between the relative position and number of contacts required.

Enabling an Application with Gestures

To enable an application with point-based gestures, software must store and evaluate changes in the parameters of contacts over time. Gesture development should be highly iterative and driven by two key parameters: the usability of the gesture set by the application's target users, and how reliably the system can provide data to detect the gestures.

For each of these, a highly iterative design process is encouraged. Limitations in the usability of the gestures are best discovered by a user-centered design process, including studies of the initial discoverability and learning time for each gesture. System limitations are also best discovered through a cycle of implementation and testing. Potential system limitations include the frame rate of the system (how often the current position of each contact is reported,) and the maximum velocity a contact can travel before it is lost and re-acquired by the system. Both of these are being improved with each release of the Surface platform.

5.3.2 Enabled Gestures

Microsoft Surface currently detects and responds to a small set of gestures. Future releases plan to expand this set. At present, some gestures are handled automatically, while others are provided in a framework which allows customization of the system response.

Touch and Hold



Definition

This gesture (also known as press and hold) is achieved by touching a point on Surface and not moving the contact area for a given length of time.

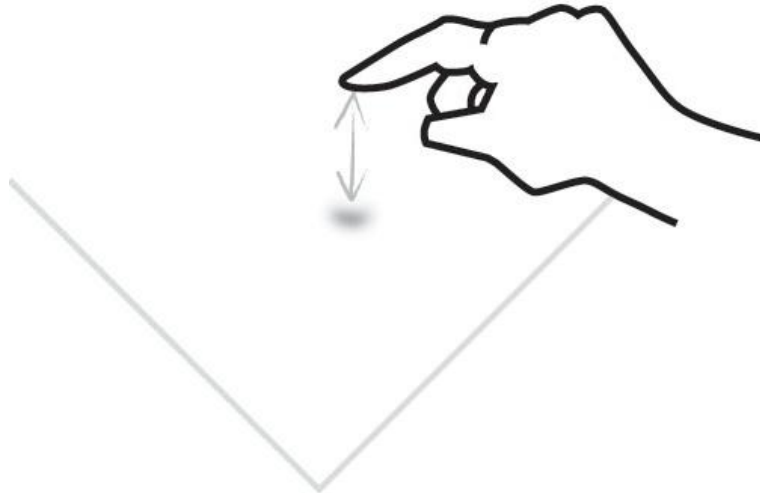
Scope

Controls in the Surface SDK can be enabled to respond to a touch-and-hold gesture. The response can be dictated by the application designer.

Utilization

Generally, time-based interactions are discouraged. They tend to be difficult for the user to discover, and the efficiency of an input sequence is limited to the timing of the gesture. As a result, it is important to limit use of this gesture to rare interactions only completed by expert users.

Tap



Definition

This gesture is achieved by quickly touching and then removing a finger from a point on Surface.

Scope

Controls in the Surface SDK can be enabled to respond to a tap gesture. The response can be dictated by the application designer.

Utilization

Users can easily discover the tap, likely because of its similarity to a mouse click. Applications on Surface use this gesture regularly. In Music, for example, a tap causes the album cover to flip, revealing the track list.

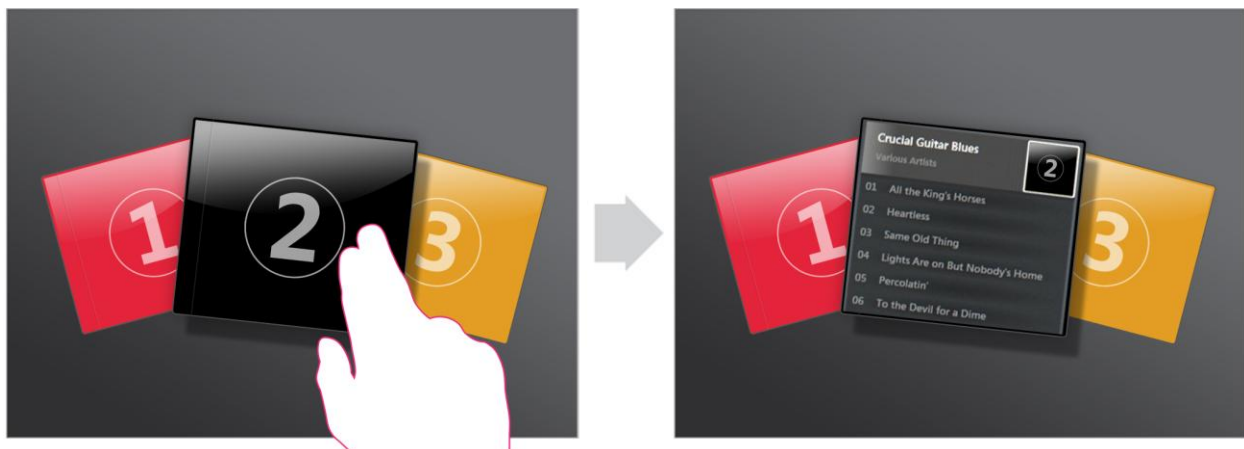
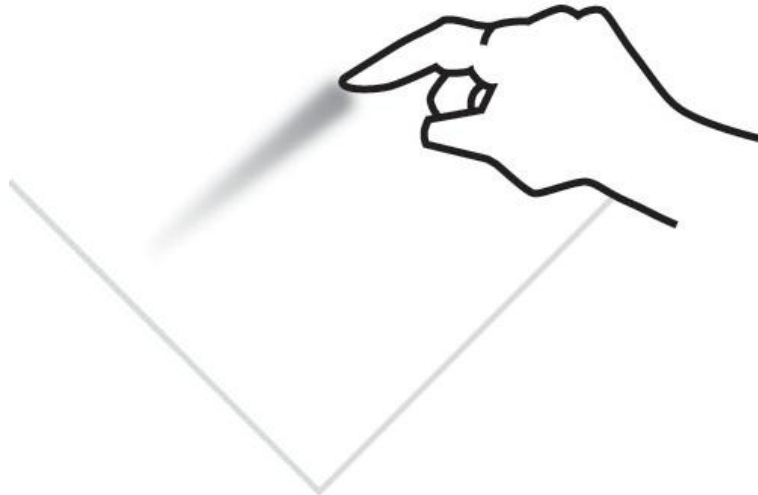


Figure 5.2: In Music, a tap on the album cover flips it to reveal the track list.

The ideal use of the tap gesture is for direct and physical action to on-screen objects. In the above

example, the user's tap can be thought of as providing a physical push to the album, causing it to flip over. Although this gesture can be mapped to non-physical actions, such uses are discouraged.

One-Finger Drag



Definition

This gesture is achieved by touching a single point of an object and sliding the contact across Surface.

Scope

Some controls in the Surface SDK can be enabled to respond to a one-finger drag gesture. The response is dictated by the controls and cannot be customized.

Utilization

One-finger drag is typically utilized to move an object from one position to another. The response of the control being dragged depends on the constraints imposed on that control. When dragging inside of SurfaceListBox, for example, the list scrolls in one dimension. The amount of scrolling is proportional to the size of the movement along the same dimension as the list.

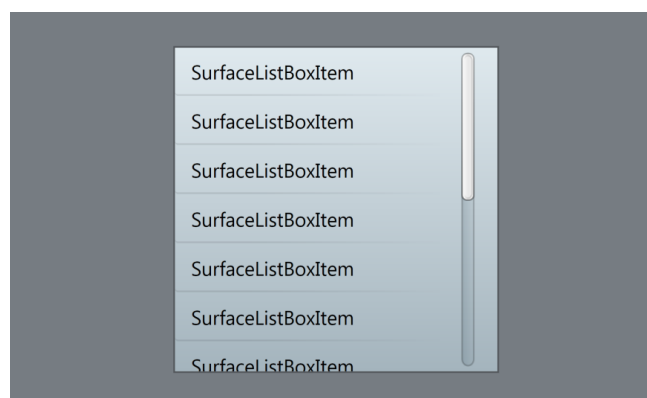


Figure 5.3: The user can utilize a single-finger drag gesture to scroll a SurfaceListBox. The movement of the list is proportional to the movement of the finger along the same dimension as the SurfaceListBox.

Unconstrained single-finger dragging is also possible. When enabled for two-dimensional scrolling, the content of the SurfaceScrollViewer moves in two dimensions proportionally to the movement of the finger. Single-finger dragging can also be enabled to simultaneously control three parameters of an item: ScatterViewItem controls can be simultaneously moved and rotated with a single-finger drag.

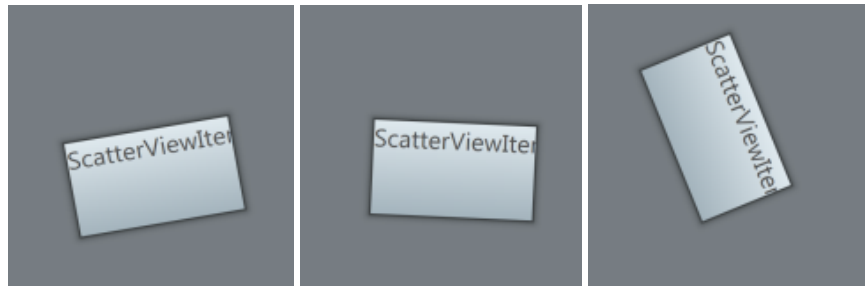
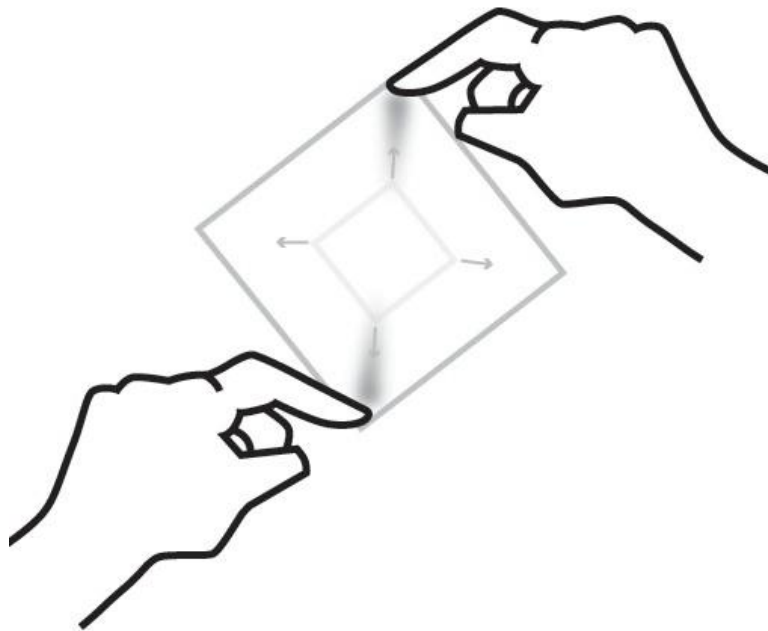


Figure 5.4: A one-finger drag on a ScatterViewItem causes it to move proportionally to the movement of the finger. In addition, the ScatterViewItem rotates beneath the finger. The amount of rotation is a function of the distance of the touch from the center of the ScatterViewItem.

The algorithm for performing a single-finger drag is internal to Surface, and not customizable. In general, the algorithm uses the distance from the finger to the center of the object to determine the amount of rotation when doing a single-finger drag. Users require an intuitive method for moving an object without rotation. To that end, the center of the object contains a zone where a single-finger drag does not cause rotation.

Multi-Finger Drag



Definition

This gesture is achieved by touching two or more points of an object and sliding one or more of the contacts across Surface.

Scope

Some controls in the Surface SDK can be enabled to respond to a multi-finger drag gesture. The response is dictated by the controls and cannot be customized.

Utilization

Response to multi-finger drag is a signature of the Surface experience. The appropriate response to a multi-finger drag is typified by the ScatterViewItem: when two contacts are dragged on a ScatterViewItem, it rotates, moves, and scales to keep the contacts above the same points of the item. When more than two fingers touch the ScatterViewItem, an average is performed so that the relative positions of the contacts and points are maintained.

5.3.2.1.1.1 Item Reaction: Zoom and Enlarge

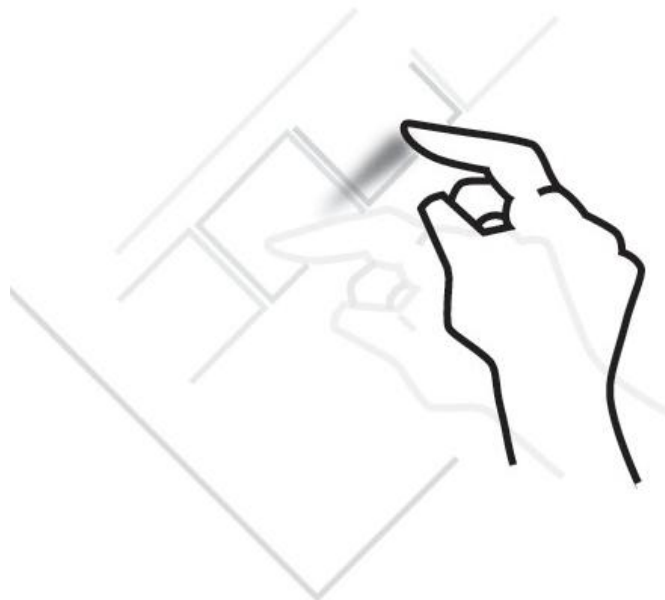
An important consideration in the multi-finger drag is the distinction between zooming and enlarging the item. When an item is *zoomed*, all its elements increase in scale so that they appear larger. When an item is *enlarged*, it increases in size, but elements within it remain in the same scale. An example is shown below:



Figure 5.5: A two finger drag on a ScatterViewItem causes it to increase in size. Its contents react differently: the photo zooms, while the text box enlarges. Determining which your object should undergo is an important design consideration.

In most cases, objects have an obvious response to either zooming or enlargement: a photo under zoom increases in size, while under enlargement, its crop is changed. A SurfaceTextBox under zoom causes its font size to change, while under enlargement, its font size remains the same and the area for entering text is increased. Determining whether content should respond to multi-finger drag by zooming or enlarging depends on user expectations for the appropriate response, and on how well that response maps to the most common tasks. Surface SDK controls have default behaviors which map to the most commonly appropriate responses.

Flick



Definition

This gesture is achieved by touching a point on Surface, quickly sliding the finger in a short line, and then releasing the finger.

Scope

Some controls in the Surface SDK can be enabled to respond to a flick gesture. The response is dictated by the controls and cannot be customized.

Utilization

Surface SDK controls with continuous parameters utilize the flick gesture to quickly increase or decrease the value of a parameter. The SurfaceThumb on a SurfaceSlider, for example, can be quickly moved towards one end of the track using a flick.



Figure 5.6: The user can quickly set the SurfaceSlider value to 100% by performing a flick gesture. Left: the user captures the SurfaceThumb by touching it. Right: the user has flicked the SurfaceThumb, causing it to quickly increase in value.

Movement in response to a flick is dictated by an inertial processor. When the flick is first performed, the value of the parameter quickly increases. After a short time, the velocity of the change decreases, as if slowed down by friction. The movement then comes to a halt. Response to flick gestures can be enabled and disabled for each Surface SDK control which allows it.

6.0 Visual Design



While interaction design defines behaviors, gestures, and responses, it is visual design that brings these elements to life on-screen. Designing visuals for Microsoft Surface, and the NUI paradigm, differs from other traditional computing experiences. The final on-screen visual experience is a major factor in creating user satisfaction, and a key opportunity to cement an emotional bond between the user and Surface.

The key to successful visual design for Surface experiences is a design language that subtly and subconsciously teaches the user. They can visually see where to touch, drag, flick, and more without explicit instruction or traditional GUI-based interface elements. This is certainly a balancing act, as visual design should add beauty and branding to the experience, but should never distract from the content being shown. However, these principles also provide wonderful opportunities to build emotional connections between users and brands, as well as between multiple users. By focusing on solid design fundamentals, any designer can create the best Surface experiences possible.

6.1 Approach

6.1.1 Creating Visual Design Vocabularies

Designing for Surface, like any other human-computer interface, is best done by creating an extensible design vocabulary, a language of shapes, forms, colors, and controls that will help visually guide users through tasks to meet their goals. These vocabularies should exhibit several key characteristics.

Consistency

Visuals must be consistent if they are to help the user find his or her way through a Surface experience. That is not to say that every application or experience has to look the same. However, within one application or set of controls there should be a standard set of forms, colors, shapes, textures, and other design elements that will help users orient themselves and anticipate what will happen when they touch something. Strong cues should be taken from the existing Surface look and feel, as seen in Launcher and other Surface applications. This allows users to transfer what they have learned from interacting with other applications and apply it to any Surface experience they have. Visual consistency makes Surface experiences predictable, and therefore repeatable, shortening the user's learning curve.

Flexibility

Visual design vocabularies should be extensible and flexible. Most Surface content and controls can be freely rotated and scaled, so a rigid and inflexible design vocabulary will not be able to accommodate such changes in size and shape. A design vocabulary should be able to be extended from one control to another, so that different controls will look like they all belong to the same "family" of objects. It is important to keep one's design flexible and extensible to ensure that all states of all objects look related and harmonious.

Premium Experience

Attention to the finest of details can help deliver a top-value, "premium" experience. Visual design should be seamless, a core design pillar of the Surface experience (see [Design Principles](#)), helping suspend disbelief and immersing users into a magical, yet self-consistent, environment. Every piece of content,

every control, and every change in application state should be supported by logical, predictable, and beautiful visuals to maintain the user's context and the continuity of his or her experience. Attention to detail does not mean adding superfluous detailing, but rather making sure that those visual details that do exist are highly considered and finely crafted.

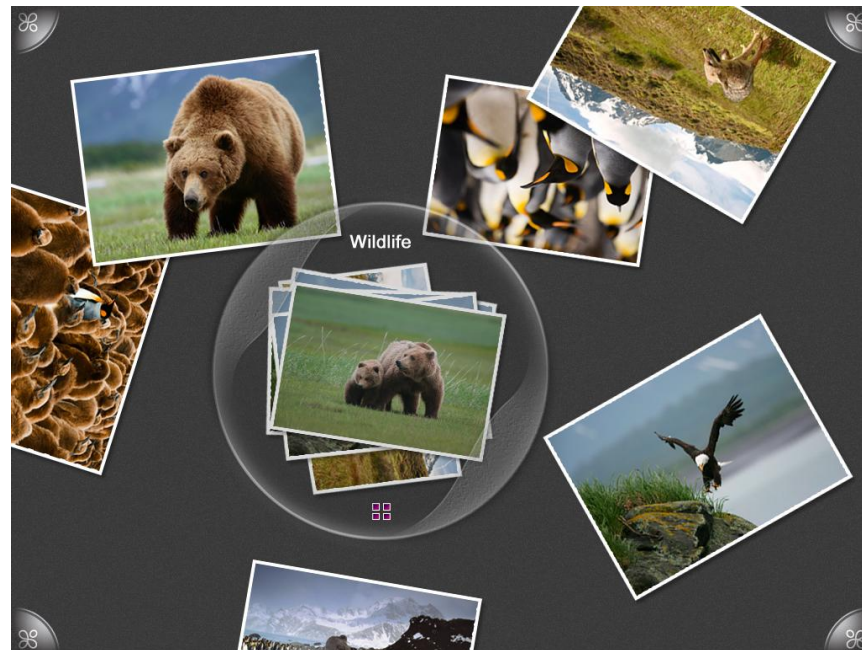


Figure 6.1: A fine attention to detail, even on the simplest and most minimal controls, creates a premium experience with maximum impact.

6.1.2 Visual Design Attributes

The best-designed Surface experiences share a handful of high-level visual attributes.

Understated

Microsoft Surface experiences are typified by their lack of visual noise and clutter. It is important to find ways to visually say the most by speaking the least. The sense of wonder and surprise in Surface comes from the aggregate user experience, not the beauty of any discrete visual element. The design must support the content and be beautiful, but without calling attention to itself. Like the best music in motion pictures, the best Surface designs are supportive but nearly invisible, although their presence is critical, and their absence would be disastrous. Utilizing tasteful restraint is a great way to arrive at understated elegance.

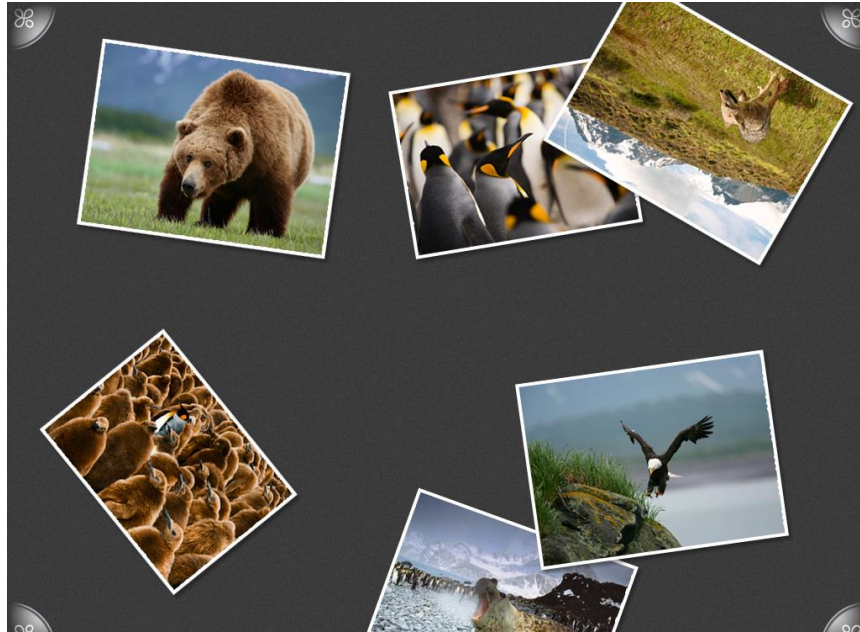


Figure 6.2: These photos scattered on a Surface screen are a perfect metonym for direct and immediate interaction. The content is the interface, one of the most important principles in designing visuals for Surface.

Minimal

In a Surface experience, content should be the interface itself, so it is important to keep visual elements to a minimum. A “less is more” aesthetic goes a long way towards this goal. Ornamentation and detailing should only support the discovery of on-screen controls and functions, and perhaps some clue as to their function. Eliminating all unnecessary ornamentation, excessive detailing, and needless controls will let the content take center stage at all times. Minimal doesn’t mean small, but rather elegant and simple, so be sure to maintain a proper sense of scale to preserve text legibility, the user’s context, and their sense of place in the experience on the whole. Microsoft Surface should be thought of as a stage upon which users act to create their own stories, like improvisational theater; users need space to express themselves and not be overwhelmed or distracted by “scenery.”

Welcoming

Approachable, discoverable, forgiving, and exploratory: these are all attributes of Surface experiences that can be heavily influenced by visual design. Colors and forms should be warm and welcoming, and erroneous or mistaken touches should never be met with jarring results or the perception of failure. There should be no fear or exclusivity in a Surface experience, but instead provide an environment that is welcoming, inclusive, and rewarding.



Figure 6.3: The Music application uses CD covers (abstracted here for copyright reasons) for direct manipulation, and a friendly and approachable virtual device for playback, to create a metonym for jukebox-style functionality.

Narrative

A Surface visual experience must support two levels of storytelling: how an experience can be explored, and the actual exploration itself. A Surface experience must be visually unified and somewhat self-explanatory; while a user might have a little learning to do in order to achieve a result, the visuals must always let that user (and other users who might simply be watching) repeat that result in a way that is both self-evident and consistent. The visuals should lead the user, but then be ready to let the user go to create their own path, experience, and narrative. Microsoft Surface users should be making connections amongst themselves and telling their own stories, and being social, one of the key principles of the Surface experience (see [Design Principles](#)). Always keep in mind the action happening “above the tabletop” and create visuals that guide at first, and then recede into the background once learned.

6.2 Guidelines

6.2.1 Visual Branding Principles

Extending a brand to the Surface experience is a bit different than extending brands to the Web, broadcast, or other media. This is because *the experience itself is the brand*. The elegance and ease of the Surface experience should be closely associated with the business and brand that is offering it; it is through having a Surface experience that a consumer or user can gain a positive brand impression. Ensuring that the visuals support such an experience is the primary goal when visually extending a brand to Surface.

Microsoft Branding Requirements

There are certain aspects of the Surface experience that should never be modified, in order to retain predictability of interactions and user satisfaction. These requirements, as well as guidelines for signage, co-branding, and more, are fully described in the [Microsoft Surface Identity Guidelines](#)⁵.

Customer Branding Opportunities

If the best Surface interfaces are nearly invisible, how are brands best visually represented? Following the visual design attributes mentioned earlier (see [Visual Design Attributes](#)) still offers the best advice: tread lightly by being minimal, understated, and welcoming. Users should not feel like they are being exposed to advertising, which is a pitfall of “over-designing” or excessively branding the Surface user interface.

Remember that the visual design of Surface software is not the only tool available for brand extension. Subtle visual branding mixed with other techniques can have big impacts. How things move, created through interaction and motion design, can convey key brand descriptors (such as “agile,” “reliable,” or “stable”) better than static visual design. The ease of which the experience flows and unfolds before the user itself can convey an extremely positive brand impression. Audio design offers a huge opportunity for adding pleasant surprises and emotional experiences. The environment in which Surface exists can also have a lot of impact, from signage to seating.

Always keep these elements in mind to achieve a highly branded *aggregate* experience. This will help to keep the Surface user interface elegant, clean, and welcoming.

6.2.2 Layout

Laying out visual elements on Surface has some interesting challenges, given the 360° nature of the multi-user interface. Any user from any side should be able to read, understand, and interact with any object on screen at any time.

There is no true *absolute direction* within the Surface user interface; that is, there is no explicit top, bottom, left, or right. Instead, it is best to think in terms of *relative position* to each user, wherein an object might simply be rotated or placed towards, or away from, a user.

On-screen layout should keep these principles in mind in order to make sure that on-screen objects are not only placed and oriented correctly by default, but easily recognizable from any angle. If an object is movable, its form and design should indicate to any user, on any side of Surface, that the object can be freely moved and oriented as the user sees fit.

Application Orientation

When an application is launched, it is important to ensure that its default orientation is beneficial to most users around it. There are generally two ways to do this.

1. If an application must be facing in one particular direction, it should orient towards the Access Point that launched it, or follow the same orientation as Launcher. For example, applications that require immediate text entry, such as logging in, will need to be explicitly oriented in order to facilitate easy access to the relevant controls (such as a virtual keyboard).

5

<https://brandtools.partners.extranet.microsoft.com/Corporate/Guidelines/Product+guidelines/Microsoft+Surface+identity+guidelines.htm>



Figure 6.4: Microsoft Surface first orients Launcher based on which Access Point has been pressed (towards either side of the screen). Then, any application launched will also be oriented in the same way by default. This ensures continuity and proper orientation for a sequence of touch inputs from a single user.

2. If an application can have its objects freely oriented, rather than the entire on-screen structure needing to face one direction, then the orientation of those objects can, and should, be predetermined in a sensible way, even if the user can change these orientations later. For example, if photos are to be spilled over the Surface towards all users, some photos should be oriented properly towards each side of the screen. This allows anyone on any side of the screen to at least see some photos properly oriented, while placing no restrictions on what they can do with them after initially appearing.



Figure 6.5: While the Photos application seems to arbitrarily distribute photos across the screen, careful inspection reveals that the photos near the edges of the screen are oriented in such a way that any user will at least have some photos that are “right-side up.”

Gridless Layouts

Most visual designers have learned to create layouts based on grids; this is one of the most fundamental graphic design principles. The 360° nature of Surface, however, is conducive to laying out applications without a screen-wide grid system, requiring a fresh perspective on visual layout.

The ScatterView control (see [User Interface Controls](#)) is a natural example of one method for achieving gridless screen-wide layout. Its very nature encourages user-driven organization and exploration of content. Acting as a metonym for objects simply sitting on a tabletop, ScatterView is natural, direct, immediate, and even a bit playful (being quite unlike work-oriented computing experiences), all of which encourage curiosity and exploration.



Figure 6.6: The ScatterView control is the perfect example of a Surface-appropriate gridless layout scheme that encourages direct and natural manipulation.

However, “gridless” does not always mean “arbitrary.” Some affordances should always be made to orient some content towards each user. The ScatterView control, for example, allows for some content to be oriented towards each edge of the screen by default.

Layouts need not be gridless all the time, either. For example, the Photos application makes use of the Scroller control (see [User Interface Controls](#)) to let the user enforce an order on content so that it may be sorted, filtered, and organized. Therefore, it is possible to use layout grids modally, letting users switch between different visual organization methods as they wish.

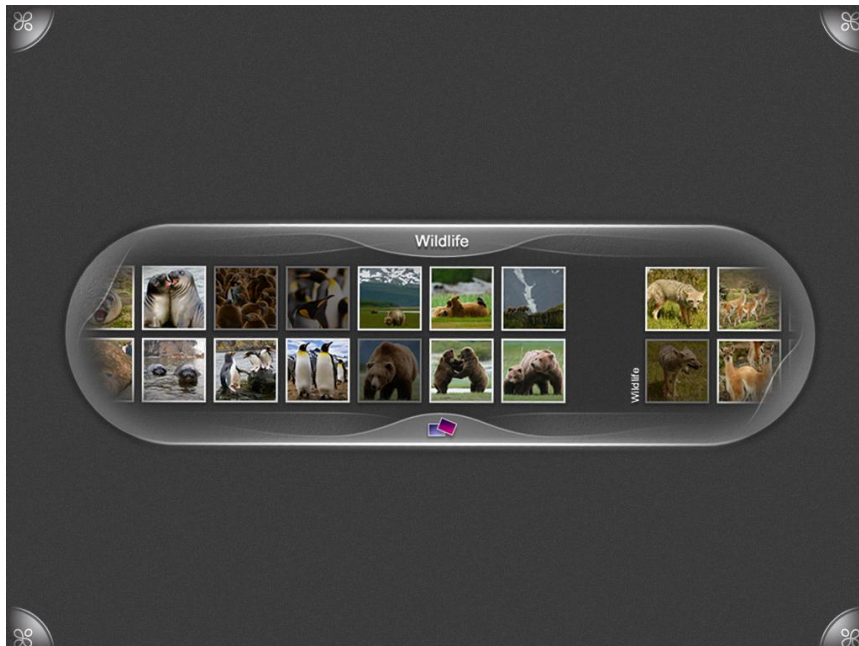


Figure 6.7: The Scroller control lets users enforce order upon on-screen objects, allowing for meaningful switching between gridless and grid-based layouts based on the user's needs or context.

Gridded Layouts

Despite the multi-user, 360° Surface experience, Grid-based layouts still have their place and can be incredibly useful. Gridded layouts are ideal for productivity or focused activities, linear sorting of data, or simply to create a visual rhythm to certain screen states. Layout grids for Surface experiences can be thought of as being either *global* (screen-wide) or *local* (within an object, piece of content, or a control).

Global layout grids determine the arrangement of content and controls screen-wide, which can be a double-edged sword. While maintaining a structure conducive to visual rhythm and easy scanning, they sometimes force the entire interface to be oriented towards one side of the screen. This can make it difficult for other users to collaborate or contribute to a Surface experience, since the interface might be easily legible and accessible from only one side. Giving careful consideration to content and control orientation for all users, on all sides, is critical in laying out elements on a global grid.

Global grids are also useful when duplicate controls should be offered to each user, or to each side of the Surface screen. This is most important for *concurrent, collaborative experiences* in which multiple users will need their own sets of controls, to either facilitate real time interactions or prevent physical interference of multiple users trying to access a single control. For example, if multiple users were playing a card game, each user should have not just his or her own hand of cards, but controls for drawing, discarding, or requesting a new deal. However, collaborative action that needn't be concurrent still does best with only one set of controls. For example, users could collaborate selecting songs for a playlist in a "jukebox" music application, but having multiple controls for adding to playlists would be confusing to not only the user, but the application itself, as the playlist attempts to sort out who made which addition in what order.

Local grids are layout systems that apply to specific objects, not the screen as a whole. They are what give structure to controls and content, making them usable and easy to visually scan. Controls and content tend to be most usable from one orientation, such as a list view control, or any content that is primarily text-based (and needs to be read in a certain orientation). Therefore, it is important ensure that they remain floating and draggable in order to be rotated easily to the right orientation.



Figure 6.8: While the layout of the Concierge application is freeform and gridless, each element is carefully composed with a local grid to facilitate visual scanning and to create a rhythm of information from object to object.

Guidelines

- Always look for opportunities to move away from global gridded layouts in order to ensure that applications are equally usable and readable from all sides, and therefore by any user.
- Plan default object locations and orientations for a 360° multi-user experience; each side of the screen should have content facing it.
- Allow users to freely switch from less-structured content viewing (using controls like ScatterView) to highly structured content viewing (by using controls such as Stacks and Scrollers). In addition, provide ways for users to reset gridless layouts to their default states; should the user lose their sense of place in a freeform, gridless experience, they should have an easy way to regain their bearings and continue.
- Always consider multiple users on all sides of the screen when designing with a global grid system; what's perfect for one user, or one orientation, may make an application difficult for others to use, reducing opportunities for user collaboration.
- Use local grid systems within (and between) controls and content so that each on-screen object shares similar visual spacing and rhythms. Ensure such controls have obvious affordances for dragging, rotating, and scaling (as appropriate).
- Look to other media for layout inspirations. Photography's "[Rule of Thirds](#)," for example, can help add interest and drama to full-screen layouts that are not specific to any particular orientation).

6.2.3 Depth

Surface is inherently a spatial experience along the width and height of the screen—the x-axis and y-axis. Using the z-axis, or depth, for content organization and visual prioritization can be incredibly powerful.

Depth helps separate foreground elements from the background, giving the user clarity around what to touch. It helps clarify what content or controls have focus or prominence. Depth can also be used to show a sorting order or represent a sequence of content.

The visual depth of Surface experiences should be “shallow,” rather than “deep.” It is important to not make the user feel that the enclosure itself is a portal or a window through the floor, but rather that the touchable surface has just a bit of depth, enough to comfortably fit all the content and controls shown on-screen.

Depth can be represented using two techniques: two-and-a-half dimensions (2.5D), and three-dimensions (3D).

Creating Depth Using 2.5D

2.5D is typified by using techniques that simulate depth without actually requiring 3D geometries. These techniques include drop shadows, transparency, depth cueing, and apparent light sources (*see below*). This can be easily achieved without specialized 3D tools, authored in [XAML](#), and delivered using Microsoft [Windows Presentation Foundation](#) (WPF) in a highly performant way.

Scale can be used to simulate depth, especially when combined with other techniques. Scaling is an ideal technique to register successful touch input; a slight increase in scale gives the impression of an object being grabbed and moved slightly closer to the user. Large differences in scale between different levels of depths can lead to an appearance of an extremely deep background, so using scale to convey depth is best done in a subtle way.

Drop shadows are an easy and highly performant way to indicate depth, but it is important to not make their presence too heavy-handed. Dark shadows can obscure content below, and very large offsets can make objects feel disconnected from the Surface tabletop. Drop shadows are great for conveying depth relationships between large numbers of on-screen objects.



Figure 6.9: Using drop shadows is an easy and effective way to convey depth changes in response to user input.

Transparency can be a powerful way to show depth by allowing objects on lower levels to show through objects above them. Transparency is computationally expensive to render, so it is important to use it judiciously. It is a technique that is good for small numbers of objects on-screen at once.

Depth cueing uses tinting, saturation reduction, and/or contrast reduction in order to simulate distant objects, simulating atmospheric effects like fog. This can lead to an inappropriately deep sense of

distance, but using subtle reductions in brightness to convey depth and object focus can be quite effective.



Figure 6.10: This hypothetical example of a book browser shows how depth cueing can be used to separate foreground elements and convey sort order.

Creating Depth Using 3D

True 3D utilizes rendered three-dimensional geometries in real time. This enables realistic rotation of cubes, spheres, custom 3D models, and so on. 3D can be authored in XAML and delivered using WPF, but it can tax the Surface hardware and achieving high performance can be a struggle. True 3D is best created and delivered using Microsoft [XNA](#), Microsoft’s core 3D and gaming engine.

True 3D offers many opportunities for interface innovation, but use caution when designing 3D controls. Controlling a 3D object on a two-dimensional touchable surface can be difficult and confusing to the user. The details of a real 3D object—such as lighting, rendering, and specular highlights—can be overly complex for the needs of most content or data. 3D should usually be avoided for the creation of controls, as any text or labels on the 3D surfaces can become difficult to read. All this complexity can add up to an experience that is so intensely spatial that its 3D nature works against its simplicity and uniqueness.

True 3D space is best reserved for intensely immersive interactions and motion, not calling attention to realistically rendered visual design elements themselves. For example, a continuously zooming interface (maps or Microsoft Virtual Earth™ being excellent examples) use extreme amounts of 3D space for interaction and establishing a sense of spatial memory to facilitate user interaction and preserve user context...but actual 3D-rendered interface elements are practically nonexistent. Consider using depth not to describe on-screen objects, but instead as a virtual space through which to lead and guide the user.

Foreground and Background Elements

While subtle depth provides effective cues between foreground objects, such as sort order on a touch response, obvious depth provides clear separation between interactive elements and non-interactive spaces, commonly referred to as the ‘background’ of a Surface experience.

Backgrounds do not trigger events or actions when pressed, but they should remain aware and responsive

to all input, so that the user knows his or her touch has been successfully detected by Surface. Backgrounds should truly recede into the background of the user's perception, in order to subtly indicate that touches will probably not result in an action. They are best shown as flat planes that are parallel to the Surface screen, in order to retain cognitive resonance with Surface's horizontal tabletop (see [The Microsoft Surface Lighting Paradigm](#)).

The Microsoft Surface Lighting Paradigm

Lighting is a key way to convey depth between objects and the background, but it is important to understand the theory behind how all objects on the Surface platform are intended to be "lit."

The Surface experience should be evenly lit from all sides of the screen. The paradigm isn't of a light suspended over the tabletop itself, but rather a huge ring light that surrounds the screen. This is important to emulate if you use drop shadows and other techniques; drop shadow offsets are the least in the center of the screen, increasing as the object nears an edge, and the offset is always towards the center of the Surface screen.

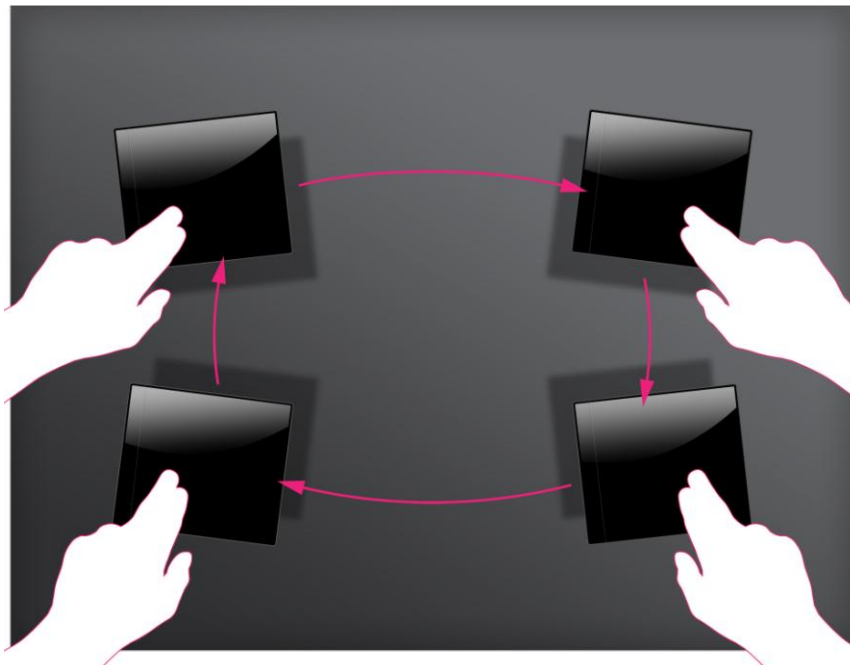


Figure 6.11: Using drop shadows is an easy and effective way to convey depth changes in response to user input.

There should never be any simulated light falloff or darkening, of objects or the background, towards the edge of the Surface screen. Remember that Surface is about superrealism, rather than physical emulation, and physically impossible perfect lighting is one such benefit of this concept.

Depth as Affordance

Depth can also be used to delineate zones for certain gestural input, to help a user understand that touching in one region will have a different result than touching in another. For example, an object's header or title bar is likely to be draggable, but a list item within that object would scroll when dragged. Making one portion of a control or object appear to be above or beneath other elements within that object can help make these zones clearer and to help users anticipate what results their gestures might produce.



Figure 6.12: This list view is separated from its surroundings using horizontal drop shadows. This helps to convey that the list may be interacted with as a sub-element without affecting the entire object (i.e., scrolling the list will not move the object, and vice-versa).

Guidelines

- Use depth to show priority, order, or focus between objects.
- Keep apparent depth somewhat shallow, not “deep” below the Surface display.
- Use depth to acknowledge successfully touched objects or controls. “Float” objects towards the user to register a successful touch. When using this technique, always reserve the highest level of depth for those objects actively being touched, so that they float above all other on-screen objects. Many existing Surface controls have this behavior built-in, and mimicking this behavior helps keep all Surface interactions discoverable, predictable, and repeatable.
- When using 2.5D techniques, it is important to remember the visual design attribute of understated (see [Visual Design Attributes](#)). Keep depth effects shallow, subtle, and elegant.
- Be aware of which depth techniques are computationally expensive in order to keep the Surface experience highly performant and responsive.
- Do not create apparent falloffs in lighting towards the edge of the Surface screen, and always orient drop shadows towards the center of the screen, as if it was surrounded by a huge ring light. All on-screen objects, and the background, should appear evenly lit at all times.
- Always carefully consider true 3D and its use. Is it appropriate or vital? If so, be innovative with its use but carefully consider its author-time complexities and its runtime performance limitations.
- Depth can be used within controls and content to delineate discrete touchable regions and help users anticipate what results their gestures might produce.
- Use transparency effects sparingly; they can be visually complex and processor-intensive. Applying transparency to solid colors is most effective in reducing visual clutter; transparencies applied to gradients requires more processing power and often makes the gradient effect difficult

to see. Use extremely subtle (5%-20%) transparency effects to bitmap or raster imagery, in order to reduce visual complexity.

6.2.4 Shape and Form

In order to let content be the interface itself, shape and form must be used to provide anticipatory clues as to the function of all Surface objects. Properly defining a shape and form vocabulary is another example of how visual design can directly impact the quality of a Surface experience.

Edges and Corners

Volumetric, organic shapes can be friendlier and more approachable than rectilinear forms, and this is a key principle in all Surface interactions and design goals. Rounded corners and flowing edges are also more conducive to interaction; they inherently appear free-floating, draggable and rotatable, encouraging users to orient controls and content as they see fit.

Straight lines and hard corners suggest rigidity and uniformity, resembling shapes users are familiar with on their personal computers. Since normal interface elements in GUI and WIMP paradigms cannot be rotated or freely oriented, this familiarity can work against the user's benefit by not suggesting the possibilities of the Surface experience.



Figure 6.13: Each of the controls above uses rounded, organic forms and extremely subtle shading in order to appear enticing, approachable, and interactive, while keeping their actual interface elements minimal yet immediately findable.

However, some metonyms will suggest the use of sharp edges and rigid lines, because their real-world equivalents share these characteristics. Examples of this include photographs and CD jewel cases, which are rectilinear in form. In this case, it is appropriate to have their Surface equivalents share these visual characteristics, with possibly pleasantly surprising superrealist enhancements and features.



Figure 6.14: Since these interface elements behave and appear like large cards, they appropriately mimic typical cards in real life, with straight edges and bright surfaces.

Consistency is vital in determining shape and form. If a control within an application has a rounded header for dragging, other draggable regions or objects should share this same look and feel.

Linework

The 360° nature of Surface requires special consideration when it comes to linework. All objects should be able to be freely oriented, rotated, scaled, and moved by the user, which can cause thin lines to appear ragged, soft, or uneven. Thicker lines look better at all possible angles, while thinner ones can develop visual artifacts like “stairstepping” at certain angles. For this reason, linework should generally be at least 2 pixels in width.

Guidelines

- Rounded forms are generally preferable for approachability, evoking organic shapes, adjusting user preconceptions, and looking more malleable than rectilinear forms.
- Ensure that interactive objects have some sense of volume, inasmuch as their metonyms will allow. For example, it would be unnecessary to have a photograph be volumetric, since its real-world counterpart has little to no volume (but shaded corners when flipping the photo over would be totally appropriate). But a critical “gadget” that has no real-world equivalent, which thus acts as a virtual device, should be made to have some appearance of volume.
- Let the form infer the function: Ensure that interactive objects look like they float, can be dragged, can be rotated, and so forth. Be consistent in the use of shape and form when developing these form-function relationships; this ensures that user gestures can be consistent, and their results can be predictable and repeatable.

- Rigid or sharp forms are generally discouraged unless the object being designed would have such forms in the real world, but do not overlook the surrealism possibilities of stretching beyond what's possible in the real world.
- Keep linework to a 2-pixel minimum width for best results when rotating or scaling.

6.2.5 Texture

At the time of this writing, texture in user interfaces has been highly refined and stylized. From operating systems to internet applications, all manner of textures and materials can be seen: brushed metal, high-gloss plastic, translucent colored glass, and more. Textures act as affordances for what can and cannot be interacted with, but sometimes act as ornamentation with no function other than stylistic differentiation.

The treatment of textures in Surface must be approached differently. Surface units inherently have a physical, real-world texture that users interact with; every touch on Surface screen provides this tactile experience. Microsoft Surface experiences should be unique, and following the conventions of the GUI and WIMP paradigms does not appropriately re-calibrate user expectations of what is possible.

The NUI paradigm of content-as-interface inherently reduces the interface "chrome" (which Edward Tufte calls "administrative debris") seen in traditional applications and operating systems, actually providing fewer interface elements that require texturing, and overuse of textures can distract from the content itself. Finally, textures on Surface should have a unique, premium sensibility that should be distinctly different than the textures seen on the traditional personal computer. Therefore, following existing examples of simulating real-world materials may not be the right approach.

Texture as Affordance

One of the most obvious uses of texture, beyond mere ornamentation, is to provide the user with clues as to what gesture is appropriate for what object, and to what that user's gesture might do. Traditional examples include ridged scroll thumbs, knurled draggable handles, and ridged pane splitters.



Figure 6.15: Traditional GUI textures help immerse or instruct the user; while appropriate for mouse-driven experiences, Surface has an inherent tabletop texture that delivers its own tactile experience.

Microsoft Surface experiences can benefit from such textures, if used sparingly, but there are other ways to let users explore and interact with controls and content. Content itself is designed to be interactive, so users will quickly realize that touching, dragging, or rotating content will result in an action, eliminating the need for texturing. Negative space within an object or control can achieve the same effect. Shape and form themselves can also suggest functionality. Brightness, form, shape, and color can be just as effective in differentiating touchable, interactive elements from non-interactive backgrounds.

Backgrounds are tempting places to utilize texture, but it is important to remember not to add too much visual interest to any element that will not generate an explicit action or result when touched. If a Surface experience was a symphony, the background would either be the softest, quietest sounds or simply the negative space between the notes. If a certain interface metonym suggests a textured background, however, that can also be an important consideration. A humorous game supposedly taking place on a picnic table (in order to reinforce its “story” or other interface concepts) certainly may need a subtle wooden-plank texture in order to retain its immersive qualities; this is also appropriate in that wooden planks are not, in the real world, inherently interactive, but rather surfaces on which other actions are performed. Even in this case, however, a realistic texture would be desaturated, darkened, and made more subtle in order to be effectively differentiated from the more volumetric and vibrant foreground elements.

Textures can help, but never at the expense of clarity, minimalism, and letting the content reign supreme. As soon as a texture calls attention to itself, it is probably too much.

Simulating Real-World Materials

Microsoft Surface experiences should not feature realistic simulations of real-world materials because the user’s sense of touch will not vary based on what textures are shown on-screen. Users will have a physical, real-world tactile experience simply by touching Surface. The Surface texture will not feel any different if a user touches something that looks like metal, or sandpaper, or glass. Since the Surface texture is consistent in the real world, creating visually smooth or rough textures creates cognitive dissonance and works against the user’s suspension of disbelief.

Emulating materials is a common design technique in traditional computing environments: chrome, plastic, and glass textures are regularly used to differentiate what is and isn’t clickable in a GUI. Calling attention to such simulated materials can be a useful affordance on the desktop, but on-screen objects on Surface should inherently be touchable and interactive. The interface should be the content itself, which users will want to touch, drill into, and interact with. Microsoft Surface experiences should reach beyond these material metaphors and instead reach a level of having their own unique, signature appearance. This is a challenging opportunity for designers to move beyond GUI clichés and create new and innovative interfaces.



Figure 6.16: The playlist controller in the Music application has an incredibly minimal texture: it looks solid, interactive, and touchable, but its texture doesn't conflict with the tactile sensation of the Surface tabletop and doesn't distract from the content-driven user experience.

Guidelines

- Remember that textures, like all other Surface elements, can be freely rotated and scaled. It is critical to develop textures that will remain elegant and understated when scaled up dramatically or rotated at odd angles. Fine linework or repeated geometrical patterns often do not maintain their elegant appearances when manipulated in this way. Be sure to test any textures on the Surface platform to see how they scale and rotate.
- While textures can act as user affordances, many other visual design techniques can achieve the same goal: negative space, form, shape, color, and more. Explore alternatives to creating repeating textures.
- When textures are used, keep them minimal and consistent, and always stay focused on encouraging touch and user exploration.
- The clarity and minimalism of buttons can be compromised if they are rendered as 2.5D or 3D elements. This isn't a hard-and-fast rule, but generally the more detailed or highly rendered (i.e., lots of shadowing, gloss, and specularities) a button is, the more overwrought it becomes with unnecessary detail. As with any other on-screen element in Surface, tasteful restraint and minimalism should be kept in mind at all times.

6.2.6 Colors

There are no explicit rules around color usage in Surface experiences; designers can and should follow good general color usage practices, as well as those rules imposed by the brands they are handling. There are, however, some guidelines that should be followed in order to optimize color reproduction for Surface's unique display system.

Microsoft Surface utilizes a short-throw, rear-projection screen inside the enclosure. Light is therefore beamed straight upwards and onto a diffusion screen, which is quite different from a LCD, CRT, or plasma screen. In addition, the vision system must be able to "see" through both the screen and the projected light in order to sense touch input and recognize objects. This impacts the final output color gamut and how colors are represented to the end user.

Color Gamut and Contrast

Due to this combination of rear projection, diffusion, and vision, the gamut of colors that Surface can reproduce is less than a desktop monitor. Its gamut generally falls somewhere between NTSC broadcast legal colors and offset color printing. Therefore, it is essential to proof all colors and designs on Surface itself before development begins, and to not rely on one's own desktop monitor for proofing.

Both brightness and color perception are altered by the Surface screen. The brightness gamut is the most noticeable constraint. Pure white has a tendency to "bloom," reducing sharpness and clarity, and pure black renders as a very dark gray. Because the rear projection system is projecting light upward towards the user, large areas of bright or saturated colors can create optical discomfort for users. Cooler colors tend to lack the intensity of warmer hues on the Surface screen, coinciding with the ability warm colors have to entice users and put them at ease.

Guidelines

- Proof all colors on Surface during the design process; proofing and iteration is the best way to achieve consistency and optimize brightness and color for the Surface screens.
- Design for a brightness range of 90% gray (near white) and 15% gray (near black) to reduce blooming and improve text legibility.
- Bright or saturated tones are best used in smaller regions to reduce eye fatigue. Darker, muted backgrounds will allow brighter content and controls to take the prominence they need to be usable.
- Warmer colors retain their intensity quite well, and reinforce a welcoming and inviting tone better than cooler hues. If this approach is supported by one's brand, it can help Surface experiences to stay engaging and inclusive.
- Avoid light from spilling onto the Surface display, whether it is from bright ambient light or nearby incidental lighting. Just like a movie screen in a theater, light spill will reduce apparent contrast and saturation. Microsoft Surface displays in dim environments have a very pleasing, rich range of tonal reproduction.
- High contrast, such as pure black on pure white, reduces the effectiveness of anti-aliasing (i.e., the subtle edge blending of foreground elements onto the background), making text look harsh and introducing "stairstepping" into object edges when rotated.
- Treat the background as a stage against which all action occurs; it should be the calmest, understated, and de-focused area of the experience.

- Subtle gradients bear careful attention on the Surface display; the reduced color contrast may cause subtle gradients to “flatten” and appear as one continuous tone.
- Be willing to experiment, iterate, and explore as you become familiar with the Surface display characteristics. Keep an open mind towards taking fresh approaches to a brand’s established color palette.

6.2.7 Typography

Microsoft Surface has a number of aspects that make its on-screen reading experience quite unique. With content acting as the interface, and using minimal (sometimes invisible) user interface elements, on-screen text has even more opportunities to act as the interface itself. Content and controls must be equally legible from any angle and from all sides, including upside down; unlike traditional computing experiences, a reader’s point of view is not guaranteed to be from the same orientation. On-screen legibility on Surface is even more important than other computing platforms.



Figure 6.17: Bold, simple typography with classic sans-serif typefaces ensures easy legibility for all users, from all angles, and at most sizes.

Choosing the Right Typeface

Overall, typeface choices must be made with free rotation, off-axis legibility, and arbitrary orientation in mind. This generally results in serif typefaces being very difficult to read in a typical Surface experience. Sans serif fonts with minimal flourishes, flowing outlines, consistent widths, and generous negative space can be very successful on the Surface platform. Like any other on-screen element, typography exists to support the gestalt experience; fonts should not call attention to themselves, but rather simply feel “right” within the context of the experience as a whole.

The Surface team has had great success using both classic and modern sans serif typefaces. Arial, Franklin Gothic, Frutiger, Helvetica, Microsoft Segoe®, and Univers are all examples of font families that render very well on the Surface platform.

Guidelines

- Evaluate text rendering at different angles, before and during testing on Surface. If legibility issues crop up, there are many options for fixing them, listed below.
- Specify all font sizes in pixels, rather than points or ems. This helps to guarantee that font elements will register to full pixel rows at runtime, increasing legibility on the horizontal axis as well as preserving enough visual data to rotate smoothly and retain legibility.
- Fonts should be kept at or above 12 pixels in height at all times. No font should ever be typeset below 10 pixels in any circumstance. Scaling fonts up slightly can help improve legibility in almost all cases.
- Microsoft Surface never raises its voice to users; it attracts and engages, and it is responsive and natural. All caps and small caps should be avoided for on-screen typesetting. Title case is more natural, easier to read, and never strikes an aggressive tone.
- If serif fonts are an integral part of a visual brand experience, use them only above 20 pixels in height for maximum legibility.
- Typesetting in high-contrast reduces legibility. White text on a black background can create bleed, harsh edges, and a poor reading experience.
- Text on curved paths should be avoided if possible. If a screen design suggests that text on a curved path is the best approach, be sure to scale the text large enough to be easily legible.
- Remember that most Surface objects and controls can be scaled freely by the user. Enforcing minimum scaling limits on objects is sometimes needed to retain maximum text legibility.
- As with colors, there is no substitute for testing font rendering on Surface itself. Make sure that legibility tests for controls and content are run with the ability to rotate and scale, if relevant, to get a sense for any angles or sizes at which legibility may be compromised.
- A slight, subtle increase in a font's x-height can help open up the letterforms for better reading, especially if one desires a slightly condensed appearance for reasons of fit or available space.
- Use condensed typefaces with care at small sizes; their lack of interior negative space within each glyph can sometimes cause legibility challenges. Make sure to test early and often on Surface itself to avoid any runtime readability issues.
- Legibility may be improved by duplicating a piece of text, moving it behind the original text, offsetting it by 1 pixel each horizontally and vertically, and filling it with a dark, neutral tone.
- Always look for opportunities to have on-screen copy held within some other shape, to help avoid text floating in space without a container. This also helps to ensure that text will always have a consistent color, background, or tone behind it to improve legibility in all cases.

6.2.8 Iconography

Icons are a staple of the GUI or WIMP interface paradigm. They are visual metaphors, compact illustrations that can convey a lot of direct meaning in a small amount of space (e.g., a printer icon for printing functionality, a floppy disk for saving, etc.). Once learned, they can be an effective shorthand or replacement for on-screen text.

But therein lies the core challenges of iconography: their meanings must be learned, and they represent abstract representations of real actions or real things. This means that the ability to interpret an icon is

based on education and cultural background, making the challenge of creating universally recognized and quickly understood icons incredibly difficult.

Some icons from the WIMP interface paradigm are so common as to be universal, such as the icon for Print (a printer), and search (a magnifying glass), whereas others are not only specific, but sometimes outdated. How many more years will pass before the typical icon for Save—a floppy disk—is rendered obsolete by the decades-long absence of floppy disks in everyday computing?



Figure 6.18: Typical GUI icons are effective metaphors, “shorthand” for concise representations of functionality. Surface iconography, however, needs to be bolder, more abstract, and ideally replaced by actual content.

Icons, Metaphors, and Metonyms

Icons for Surface should act as *metonyms*, which absorb complex actions or concepts into a simpler representation, than *metaphors*, which replace a thing with, or transfer meaning to, another thing. Therefore, it is most appropriate for icons to actually be content, or at least be closely related to content.

For example, rather than designing an icon that says “menu” for a restaurant application (a metaphor), it is best to design a virtual equivalent of a menu (a metonym), which the user is more likely to know how to handle: They can scale it up to “look closer”, and flick left and right to turn the pages. The concept of surrealism then also applies when they order their food: rather than pressing an “add to your order” button, they simply touch the item in the menu itself. In this way, icons and graphical representations of functionality can ensure that Surface experiences remain natural, direct, global in interpretation, and contextually relevant.

Icons as Affordances

Since icons will always remain useful for inferring functionality to the user, they can still be relevant for some Surface experiences. In an effort to create understated and minimal interface controls, the functions of buttons and other controls are not always visually obvious, and it is not always feasible to use text labels on small controls. In such cases, bold and iconic illustrations can be used as effective alternatives to textual button labels.



Figure 6.19: The map icon shown in this application is simple, bold, and very Surface-appropriate: It reveals functionality to the user without calling attention to itself. It is also held within a piece of on-screen content, better creating a direct and immersive experience.

Icons are tools for visual recognition of application state and functionality, but like any tool, they must be used to solve the right problem. On Surface, more direct methods of manipulation are used in the place of most icons. When icons do appear, they must be universal, immediately recognizable, and easily learned.

Guidelines

- The Surface experience mandates that content is the interface, and that the interface is the content. Therefore, icons are best used as affordances, providing hints or clues to the user so that they can infer what result a touch or gesture will create. Icons need to encourage exploration and to help the interface stay learnable, discoverable, and natural. Do not let the icons become the interface: That’s the job of content.
- Like typography, icons must be legible from any angle. Do not let icons become complex or overly detailed. Keep them suggestive, simple, and truly “iconic.”
- Think globally when designing iconography, and try to avoid culturally specific imagery or references that some users may lack the context to interpret. Icons must be immediately interpreted or, at worst, easily learned and remembered once it has been interacted with.
- Buttons must be sized according to the appropriate input. The touch-based input system on Surface requires input devices that are much larger than a traditional on-screen cursor; the size of buttons may need to be larger than is typical for GUI/WIMP interactions.
- Icons should be between 5/16” and 1” in diameter in order to be legible, recognizable, and touchable. This limits icon size to between 15 and 43 pixels across, further underscoring the need to design simple and easily recognizable forms.
- Icons should not illustrate, but rather should hint, guide, and infer.

6.2.9 Motion Design

Motion design defines how things move on-screen, and is a critical part of the Surface experience. It is motion design that provides the animations and effects that most powerfully convey emotion, action, interaction, system response to touch input, cues and invitations to explore, screen transitions, and much more.

Motion design is never gratuitous in a Surface experience; animations always support the content and the experience as a whole. Motion design is created in extensible vocabularies, just like visual design. Transitions are used to provide critical functionality clues and to make sense of application states. Ambient animations help to build both brand and an overall sense of personality of a Surface experience. No matter what the technique or effect, motion design should always be consciously used to create experiences that are natural, alluring, unique, and responsive.

Designing Motion Design Vocabularies

Motion design should be created as an extensible vocabulary of consistent and shared behaviors, just as a visual design vocabulary is built from a library of color, shape, and typographic rules. A motion design vocabulary should create a sense of rhythm for all movement on-screen.

Rhythm can be created by devising a system of consistent time intervals, which can give a Surface experience a palpable rhythm. For example, if objects move in intervals of 150 milliseconds, or half-seconds, pre-animated behaviors will feel consistent and predictable.

Rhythm can also be reinforced by standardizing apparent weights of objects. On-screen content and controls should feel light, floating, and responsive, with just the slightest sense of weight or inertia. Therefore, all animations should accelerate and decelerate—an animation technique known as easing—in order to convey a sense of realism. It is important to remember to use deceleration, and the edges of the screen to even artificially dampen velocity, so that no on-screen object can be flicked or thrown off-screen, never to return.

A consistent motion design vocabulary helps to make Surface interactions learnable (if animations are tied to functionality), repeatable, and predictable.

The Importance of the Transition

Most importantly, motion design governs how all transitions occur. Transitions build context and sense of place for the user throughout their entire Surface experience. They help users to build mental maps of their experiences, shows how to use controls by example, and remember where on-screen objects have gone if they are minimized or moved.

Individually, transitions are not as important as how they all work together and with less dynamic screen states. Put together, however, transitions stitch together discrete moments and actions into a seamless, natural experience that is unique, intuitive, and responsive to the user. In this way, transitions are a key building block of an ideal Surface experience.



Figure 6.20: This image sequence shows the importance of the transition: seamless, immersive experiences reduce frustration by always maintaining continuity, and helping the user maintain a sense of place and context.

While interaction design defines behaviors, gestures, and responses, it is visual design that brings these elements to life on-screen. The final on-screen visual experience is a major factor in creating user satisfaction, and a key opportunity to cement an emotional bond between user and Surface.

Latent Learning: Transitions as Affordances

U.S. behavioral psychologist Edward C. Tolman coined the term “latent learning” to describe how learning occurs passively by repeated observation. Any experience with a Natural User Interface will take some time to learn, like any other human experience, but Surface should enable extremely natural, rapid learning and reward exploration with constant enjoyment.

Motion design gives the designer many tools to allow Surface users to rapidly learn by passive observation without requiring explicit tutorials or demonstrations. For example, consider an application with a list box control. If the list box suddenly appears on-screen without any transition, it can sometimes appear deceptively static. Are there more options than are shown? Is it scrollable?

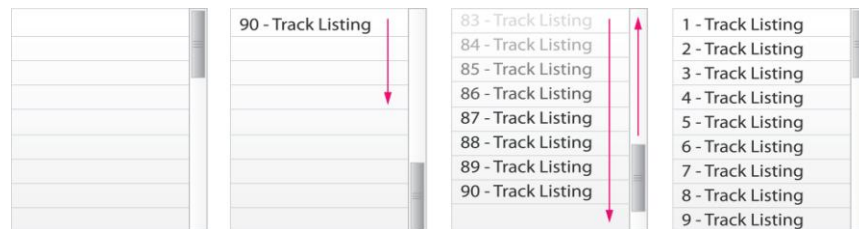


Figure 6.21: Showing a List Box’s contents slide in when the control is instantiated is a powerful way to convey a lot of information quickly. It suggests that this list is indeed scrollable, and reveals that there is more content beyond what can just be seen by default, encouraging user interaction.

Some visual affordances (see [Depth](#)) can help, but if the list box appears empty just for a moment, and then its content slides in visually, the user sees this movement and can infer that there are more options than are shown. Likewise, the sliding movement acts as a clue that the content that appears isn’t simply static, but that it can be manipulated.

This is just one example of how motion design can be used to create transitions that passively and succinctly illustrate how Surface content and controls may function.

Ambient Animations

Few computing experiences can be said to have much in the way of a personality. Sometimes this is a good thing; one generally wants to impose one's own personality, work habits, and techniques upon a piece of productivity software. However, Surface experiences are different. They should be unlike other computing experiences: Surface interactions should be magical and emotional, and with a sense that Surface has a personality and some sense of awareness.



Figure 6.22: Simply having a horizontal ambient animation gives a subtle clue that Launcher is interacted with by horizontal or side-to-side gestures. Notice the Application Preview Movie playing as well.

Let the user see his or her interactions change the state or mood of Surface itself. Subtle ambient movement in the deep background can help make a Surface experience have some life, while altering this ambient movement based on user input can give a Surface experience a personality. For example, a background animation could change in color from neutral to warm based on how many touch inputs are sensed. When several objects are placed on Surface and recognized, animations could build visual connections between the objects to illustrate relationships. Visual touch feedback could change shape, color, or rhythm based on how many touch inputs are registered within one object, suggesting that new gesture possibilities exist, and encouraging interaction. Microsoft Surface offers many opportunities for creating ambient animations that help maintain emotional engagement.

Effects

Effects can be animated over time to respond to user input; the simplest example would be drop shadows (see [Depth](#)) that change their offset and orientation based on where the user moves them. Other effects that can be used at runtime include color and brightness manipulation, blurs, glows, displacement/distortion, and more.

There are two primary challenges in using effects: visual intensity and performance.

Microsoft Surface experiences need to maintain a sense of subtlety, elegance, and restraint while still feeling aware, responsive, and engaging. A sure way to visually overwhelm the user is to use too many effects, fast movement, and overly energetic transitions. Any effects that are used should support the overall user experience, and usually the most beautiful effects are the most understated.

Effects are also very processor-intensive, and need to be used sparingly to conserve computing power. If any interaction or animation feels sluggish or unperformant, the user will sense that Surface is being unresponsive, uncooperative, and this will lead to a poor emotional connection with the Surface experience. Distortions are especially taxing and should generally be avoided, as should blurring, unless these effects are part and parcel to the content being displayed, the themes being conveyed, or the metonym being followed.

The best way to approach effects is to use them as enhancements, not critical elements. In this way, they can be more easily scaled back, altered, or removed to improve performance without compromising the visual or motion design of the experience.

Guidelines

- All on-screen objects—indeed, the entire Surface experience—should feel smooth, agile, light, and flowing.
- Motion design on Surface should never be jarring, sudden, chaotic, or intense. Microsoft Surface experiences are not about adrenalin, fast-paced movement, or visual distraction (unless, for example, these are core mechanics in a game). In fact, most of the movement on-screen will be created by the users themselves. Just as in visual design, tasteful restraint and elegant understatement will best serve the Surface user experience.
- All objects that can be moved, flicked, and thrown should ease in and out in their movements; this means that their velocity should increase at first and decrease after the user releases it. This guarantees smoother and less jarring movement.
- Easing in and out need not be symmetrical, i.e. the amount of easing in can be different than easing out. Less easing in makes objects seem to have less inertia, and therefore they feel lighter in apparent weight.
- Always establish a motion design vocabulary around recognizing, and rewarding, any user input. Even if a touch triggers no specific event, Surface must respond visually to ensure the user stays comfortable and that they know that Surface has “heard” their input. To not respond visually is to infer a failure of the user to interact with Surface, which should be avoided at all costs. This should be done for touches, gestures, and even object recognition.
- Never let the Surface experience become fully idle. Whether using ambient animations (*see above*) or interactive loading states (*see below*), a Surface experience should feel aware and responsive at all times.
- Motion design is a great way to mitigate load times or distract the user during wait states. While all Surface experiences should be sufficiently performance-tuned to minimize or eliminate all system delays, sometimes a brief lag is inevitable. This underscores the need for Surface experiences to always be reactive, inviting, and fun, even if the user simply is given a fun interactive distraction to distract from a necessary wait. Let the loading process itself be interactive.
- Motion design is a fantastic branding opportunity. Never hesitate to interpret brand descriptors or brand attributes into movement; it can invisibly build emotional connections that are inappropriate (or impossible) to convey using static imagery.
- Do not be tied to realism. Just because a sculpted, seemingly solid, volumetric object cannot be rolled or flipped like a piece of paper in the real world does not mean that this is a limitation within a Surface experience. The concept of surrealism should be used as a license to move beyond literal representations of real-world objects, and to start creating unique and magical experiences only possible with Surface.
- Animations can even be used to reduce the visual footprint of on-screen controls. For example, a scroll bar may appear only half an inch wide, but increase to be an inch wide when touched, thereby offering both a visual input response and improving the usability of the control.
- Get inspired. Watch how television commercials use motion to communicate aspects of the brands they are representing. See how the themes of the movie can be expressed in the opening and closing credits for films, especially in the classic work of [Saul Bass](#) and the contemporary work of Kyle Cooper and his firm, [Imaginary Forces](#).

- Animations can progressively disclose detail and animations in ways that help preserve user focus and context. A control or content can flow, fold, flip, or transform in reaction to user input to reveal increasing levels of detail or new perspectives on the same data. The designer's imagination is the only limit.

7.0 Physical Design



The Microsoft Surface user experience is not comprised solely of software. The Surface screen, tabletop, and enclosure are all carefully designed to support the user experience. Beyond the Surface software and hardware, the physical environment surrounding Surface also has a significant impact on user satisfaction and how brands are conveyed through the Surface experience.

That is not to say that the Surface hardware or its physical environment is more important than the software experience. Instead, the hardware and environment should be seen as stages upon which the software may act to inspire and engage, facilitating the inter-personal and social interactions of Surface users. The Surface experience is, and always will be, about the aggregate human experience, within a unified hardware and software platform, all working in concert in a physical space.

Therefore, there are some important considerations that should be kept in mind when considering alterations to the Surface hardware, and when placing Surface units into physical environments, in order to create the optimal, holistic, and unified user experience.

Remember that there are some physical aspects of Surface that must not be changed. These are listed in the [Microsoft Surface Identity Guidelines](#)⁶.

7.1 Industrial Design

7.1.1 Approach

Just as software can have an extensible design system, or vocabulary, so does the hardware and enclosure of Surface. There are aspects of the industrial design of Surface that should be kept in mind when designing Surface experiences of any kind. These are especially important when designing physical objects in the vicinity of Surface, or to the physical form of Surface itself.

6

<https://brandtools.partners.extranet.microsoft.com/Corporate/Guidelines/Product+guidelines/Microsoft+Surface+identity+guidelines.htm>



Figure 7.1: The industrial design of Surface, just like its software design, is focused on being approachable, welcoming, minimal, and unlike any computing device seen previously.

All physical elements of Surface have been designed to support the following core principles, and any physical alterations to Surface should be made with the following in mind:

- The screen is the center of attention. All aspects of the Surface experience should focus the users' attention to what is occurring on-screen. This is where all interaction occurs and where emotional connections are forged. Nothing should distract from this core aspect of the Surface experience, and all eyes should be drawn towards the Surface tabletop and screen.
- The hardware should not call attention to itself. The combination of the hardware and the software should be the attention-getter, and the software experience is what truly engages the user. Just like the user interface elements in an application, the Surface hardware and surroundings should support the natural interaction with Surface by being elegant and understated. A Surface unit should not look physically "cool," or weird, or overtly interesting in its own right, but rather simple, intriguing, and "just right." The hardware and software elements of Surface should just be aspects of one, integrated whole.
- Nothing should interrupt the 360° nature of the Surface experience. All sides should be smooth, rounded, and inviting. No protrusions should interrupt the inviting and welcoming nature of Surface from any sight line or physical direction/approach.
- Support the forward-thinking yet timeless aesthetic of Surface. Interactions with Surface should feel innovative and futuristic, revealing that natural user interactions are both possible and pleasant, but this does not mean that the hardware design should necessarily be futuristic (or "retro"). Surface units should have a timeless elegance that do not recall, or forecast, any specific era.
- Microsoft Surface should fit will into its physical environment. Surface units have been consciously designed to be color-neutral in order to blend in with almost any décor or interior design scheme. Match any color or physical Surface alterations to the environment accordingly.

7.1.2 Guidelines

When considering manipulating, augmenting, or altering the external, physical appearance of Surface units, keep the following guidelines in mind. All such designs and plans should be submitted for review to your Surface account representative.

The Physical Appearance of Microsoft Surface

- Microsoft Surface should stand out as a new kind of physical object. It is a total package of software and hardware that is redefining the human-computer interface, and defining the leading edge of the NUI and surface computing paradigm. Microsoft Surface should not recall any previous devices or eras, and should stand alone as a new way of interacting with computers and digital content.
- Take an iterative approach to any physical alterations to Surface units. Initially place Surface units in their final environments without changing their appearance. Microsoft Surface's design is meant to support a massive range of visual decors and blend in well with a neutral color scheme and translucent materials. It can be surprising how well Surface units can integrate into an environment with no visual changes whatsoever. Only after assessing its fit in its final environment should you consider physical alterations.
- Any changes must keep the users' focus on the tabletop and screen. Leading the users' eyes to this location will best facilitate interaction and emotional connections with the Surface experience.



Figure 7.2: Hardware is merely a stage upon which human interactions take place, and those interactions must always be focused on the Surface tabletop and screen.

- The Surface experience should have a sense of premium value. This is best accomplished using modern, innovative materials with positive tactile feelings, careful attention to detail, and craftsmanship.
- Consider brand and messaging when choosing colors, textures, and materials. Real-world materials can not only convey brand, but also can affect how welcoming, inviting, and intriguing a

space or object is. They can even convey certain emotions. For example, ballistic nylon conveys toughness and utilitarianism, while microfiber conveys easily maintained comfort and plushness. Choose materials and textures that fit with the rest of the hardware design language of both Surface and any branded software experiences that might be served by Surface.

- Take care with the proportions of Surface because it has been carefully designed using golden ratio proportions, a set of mathematical ratios observed in geometry and nature by the Greeks and followed in the arts since the Renaissance as a paragon of perfection (which some psychologists have validated). These help to convey a feeling of stability, timelessness, and human resonance. This is especially important if Surface units will be raised to waist height to facilitate use while standing, which radically changes its height proportion.
- Design with serviceability in mind. Easy access to all aspects of Surface hardware must be maintained, whether a custom enclosure is created or bolted-down furniture is placed nearby. Service technicians must have unimpeded access to Surface should it require maintenance.

Adding Hardware and Accessories to Microsoft Surface

Careful consideration is needed when it comes to using physical hardware accessories, such as printer, DVD drives, and more, in the Surface experience. Sometimes it is compelling, or appropriate, or necessary, for a user to put his or her own data onto Surface to manipulate. This could be medical data, photos of a vacation, or music tracks from a Microsoft Zune digital media player. Conventional wisdom would suggest that the most direct way to facilitate this would be to extend Surface with physical hardware accessories, from printers to card readers to DVD drives. However, these might not be the right approach for the overall Surface experience. Following the hardware design principles of being timeless, understated, forward-thinking, and supporting a 360° experience, some new considerations come into play.

- Consider new and contemporary ways of interacting with hardware devices, and for getting data from such devices onto Surface. Accessories such as DVD drives and media card readers are inherently, physically directional; media must be inserted from only one physical access point, and its physical controls only point in one direction. Consider more contemporary wireless means of transmitting data to and from Surface, such as Wi-Fi and Bluetooth®. Such wireless devices eliminate physical tethers, cables, and permanent accessories, and can be freely oriented to use their own interfaces as needed from any direction. This keeps the nature of Surface flexible, responsive, modern, and forward-thinking, without requiring a user to mediate data transfer between devices.
- If accessories are absolutely required for a given Surface experience, do not clutter the Surface tabletop with them. Use of the Surface screen must be an unencumbered experience, and keeping it free of objects, distractions, obstructions and accessories will make it more inviting and clearly a space in which to interact.
- Accessories themselves should have their own separate, free-standing enclosures that abstract their technical nature (such as a visually branded enclosure for an otherwise beige-colored printer) in order to be more friendly and approachable. Just like Surface itself, strike a balance between designing rugged and durable enclosures to withstand constant use and using materials that do not have a lot of visual weight to them (such as Plexiglas). Eliminating visible or exposed wires and cables is as crucial to user comfort and safety as it is to reducing the visibly technological nature of the Surface experience. Visually tying these enclosures into the hardware design language of Surface can help users understand that these elements are functionally tied to Surface.

- Microsoft Surface is not a general-purpose device; focus is one of its best qualities. Specific experiences should always carefully be crafted for Surface. Just as one would not simply port a Windows Vista® application to Surface without a significant refactoring, one should not start adding accessories to create a touch-enabled PC. The NUI principles of Surface place it far beyond a PC, and all software and hardware layers of the experience must underscore this.

7.2 Environment Design

7.2.1 Approach

The physical, tactile, and social aspects of Surface rely heavily on the surrounding physical environment to ensure success as a positive, enjoyable computing experience. Surface units must be accessible, visible, and able to draw groups of concurrent users to it without impeding access to the screen and the software experience. Microsoft Surface computing experiences are most effective when users can freely congregate together, share the experience, and connect with one another. The surrounding area must be conducive to such activities.

7.2.2 Guidelines

The following are some broad guidelines for optimizing physical spaces for Surface installations, and making accommodations for Surface users in public spaces. More detailed information can be found in the [Microsoft Surface Site Readiness Guide](#)⁷.

The Microsoft Surface Environment

- Let the environment draw users to Surface. Position Surface units near high-traffic areas, ensure clear sight lines, and use the architecture and interior layout of the space to direct users to them. Subtle use of contrast in detailing, décor, texture, color, or lighting can help set the stage for a Surface experience within a venue. This can be much more successful than direct and overt signage (see below).
- Use the environment to establish Surface as a public or private experience. While casual observation of others using Surface is how most novices learn how to use Surface, some Surface experiences may need to be intensely focused in such a way that discourages onlookers over the users' shoulders. Some uses of Surface revolve around a group of friends or family using the table in context for long periods of time, and in such cases queues should not form around these users to apply pressure to end their interaction. In other cases, ensuring a consistent and brisk throughput of users accessing Surface may be a priority, in which case the physical space should encourage onlookers behind the current users (while keeping them comfortable as well). The appropriate use of space will depend on the user, brand, and business context.

⁷ <http://community.surface.com/downloads/p/134.aspx>



Figure 7.3: Physical installations that allow room between and behind comfortable, movable seating allow onlookers to view the gestural input of the seated users before using Surface themselves, a key to learning how to use the natural Surface interface.

- Be wary of competing media or areas of interest. Install Surface units away from other sources of video, audio-visual devices, and digital entertainment content. Let Surface units have their own space to engender a social, collaborative experience between users.
- Do not install Surface units in areas of bright ambient or direct lighting. This can create glare and reduce the contrast and color intensity of the rear-projection screen. Choose a location, and then inspect the area for inordinate amounts of glare or contrast reduction, adjusting the lighting or Surface unit location as needed. The Surface vision system detects shadows or interruptions in the infrared (IR) spectrum of light, which can be fooled by direct sunlight, halogen, and incandescent lighting in the nearby environment, underscoring the importance of eliminating egregious light spill.

Signage

Physical or electronic signage can help build awareness of the presence of Surface units in a venue, and can help to drive users towards its location.

- Physical signage should aid users in identifying and locating Surface units within a venue or room, but should not offer explicit instructions on how to use Surface. Such latent instruction is best left to properly designed software experiences (see [Interaction Design](#)) which should be free of explicit instructions. Sometimes simply indicating that Surface units are in a room or venue is enough to encourage users to seek it out. Focus on building awareness of the presence of Surface; a live video projection of users interacting with Surface would be an ideal form of digital signage.
- Do not place items on top of the Surface tabletop, such as flyers, standing menus, or other potential obstructions. These items will be removed by customers in order to interact with Surface, and may end up on the floor, giving an unsightly appearance to the Surface area.

Human Factors

Initial testing has shown that the vast majority of users approach Surface units in groups of two or more, so configuring the surrounding environment to facilitate the human body can impact the overall user experience.

- Deploy Surface units at either coffee-table or waist height, based on the setting and context. The Surface tabletop is 22" tall, similar to a coffee table, which is ideal for seated users, and seated users can stay engaged with Surface experiences for prolonged amounts of time. With proper modifications to the side panels or other elements, Surface can also be easily used at waist height, which is ideal for standing users. Standing users typically have shorter interactions with Surface experiences, but this can be desirable if encouraging more frequent use by more individuals over time. The lower the Surface screen, the easier it can be seen by others standing at a distance, acting as an additional attractor for casual users or curious onlookers.
- Place moveable seating around Surface units so users may re-arrange themselves as desired. Users often engage Surface while sitting shoulder to shoulder, so allow more than one seat per side to facilitate multiple users in each orientation. Benches and couches may not be moveable, but users can easily move along them to reposition themselves to their liking, and this allows more than one user to use the same side of Surface.



Figure 7.4: Seating is critically important for users to feel welcome and comfortable with the Surface experience.

- Seating should have room for users to tuck their feet underneath themselves so that foot contact with Surface is minimized. This helps keep the side panels from becoming damaged or soiled.
- Allow seating to be moved as close as possible to minimize leaning or any other fatiguing body positions. However, try to prevent forcing users to have their feet contact the side panels.
- Ensure that users have a place to safely put down whatever they might have in their hands. Many casual Surface users will be carrying items that they will want to put down before touching the screen. While Surface units can accommodate users placing their possessions on the tabletop, this gets in the way of most interactions, and some objects such as car keys can permanently scratch Surface. Consider using small tables between movable seats or armrests that can double as

resting platforms for the user and their belongings.

Maintenance

Proper cleaning and maintenance can be vital to the success of the Surface experience. As a multi-user, multi-touch device, Surface units will inevitably become dirty with continued use, and a soiled tabletop or screen will become a significant barrier to usage. Users will not want to touch screens that are visibly dirty or soiled. While Surface units are built to withstand constant interactions, it will be necessary to establish a regular cleaning schedule to keep them attractive and welcoming. Note also that the presence of a user's feet near the side panels can result in dirt and particulate matter building up on the sides of Surface. More detailed information about Surface maintenance can be found in the printed *Microsoft Surface Warranty and Maintenance* manual.

8.0 The Microsoft Surface User Interface



8.1 User Interface Components

User interface (UI) components are a collection of applications and software subsystems that are part of the Microsoft Surface experience. Some UI components are applications in their own right, some are design patterns, and some are user tools. All UI components are more complex than UI controls (see [User Interface Controls](#)) and they usually serve a greater user experience, rather than standing alone.

UI components are customizable and this section provides design considerations and guidance to assist with design customization decisions. For step-by-step how-to style instructions to customize these various elements refer to the [Microsoft Surface Administration Guide](#)⁸.

8.1.1 Attract Application

The Attract application is meant to do just what its name implies: it attracts users, and entices them to interact with Surface. Hence, the Attract application is one of the most important aspects of the Surface experience. Users need to be assured that it is acceptable, appropriate, and even encouraged to touch the technology. Once granted this permission, the Surface experience starts to unfold. The Attract application does this without explicit written instruction, which keeps the Surface experience natural, welcoming, and direct.

The Attract application (see also [Microsoft Surface Software Environment, Attract Application](#)) can be customized in two key ways; by replacing the default Attract application with a custom Attract application, or by customizing the default Water Attract application.

Creating a new Attract application

Creating a new Attract application is harder than it might seem at first. While the final user experience should be simple, elegant, natural, and direct, the path to arrive at such a solution can be surprisingly long. The Attract experience needs to be visually compelling, appear interactive even from a distance, be emotionally engaging, and create a natural desire in users to touch the screen. It also needs to help the user discover the Access Points that lead to Launcher (see [Launcher](#)). Creating the right application to do this, while keeping it simple, elegant, natural and direct, can be challenging.

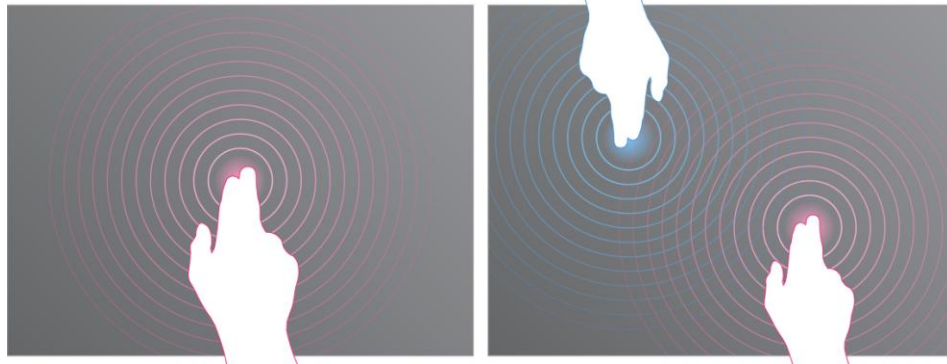
One of the best ways to describe design considerations for a custom Attract application is to look at the goals and objectives of the default Attract application, the water simulation:

- Generates curiosity and attracts users to walk over and investigate. It is approachable. It attracts, entices, and inspires, and never “yells” loudly in order to attract attention.
- Compels users to touch and interact.
- Creates a successful first touch experience with zero chance of failure. Users touching the water simulation Attract application for the first time will get an immediate response, finding it rewards, reinforces and encourages their interaction. It is instantly understandable.
- Offers a fun, engaging, and visceral experience without any chance of intimidation. There are no

⁸ <http://community.surface.com/blogs/administer.aspx>

tasks to perform, everyone already knows how to play with water, and there is no right or wrong way. It is a knockout experience that draws the user in for continued engagement, but also leaves a great impression even if the user interacts with the Attract application only.

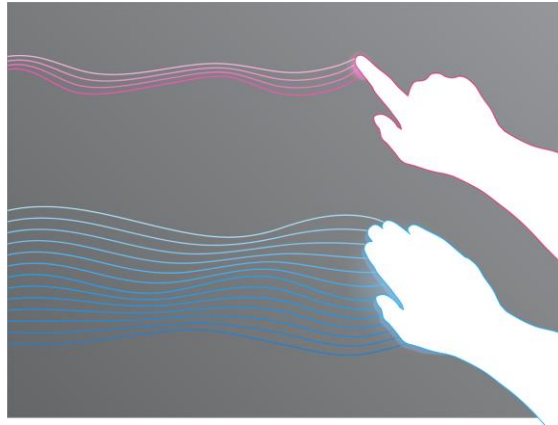
- Exposes users to Access Points and helps them learn where to find them, how to use them, and their purpose. The Access Points themselves undergo a subtle animation every now and then to get the user's attention. Since they are the only UI controls visible in the entire default Attract application, users quickly deduce that they must be touched in order for something else to happen. Once users activate them, the system navigates to Launcher, where the Access Points still remain visible and active. Through that first-time interaction, users quickly learn that the purpose of the Access Points is navigation.
- Teaches users:
 - Microsoft Surface can be approached and used from any side; it is a 360 degree user interface without any real top, bottom, left, or right.
 - Interacting with Surface is as simple as touching. Users gain confidence and gradually try more interactions... from one finger to multiple fingers. It is multi-touch.



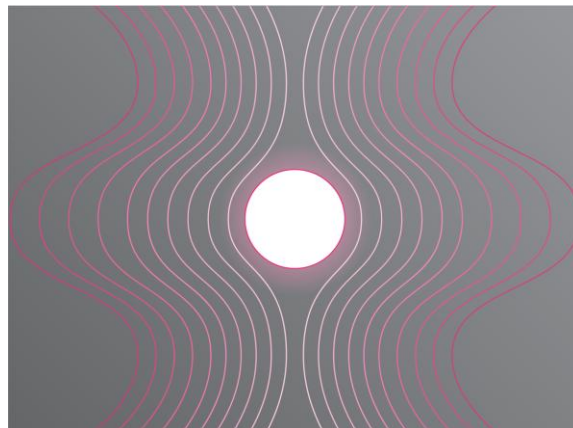
- Use an entire hand to interact with Surface, as it can sense and respond to large contacts.



- Use gestures to interact with Surface. Brush or move a finger or hand along the screen and see that it responds to more than just a stationary touch.



- Surface is social once a second or third person joins in.
- Surface responds to objects. As soon as someone intentionally or accidentally brings an object into contact with the screen, water ripples will bounce off the object.



- No explicit instructions are necessary, users can learn by experimenting and doing. After all, this is all possible with real water, and users can leverage their knowledge and experience from the real-world interactions in their own lives.

Incredibly, something as basic and elemental as water can offer further evidence that less is more. In creating a custom Attract application consider the following:

- Remember that the goal of the Attract application is to do just that, attract. It should intrigue, entice, spur curiosity, and offer a fail-proof way to interact with Surface for the first time. Its visuals and motion should not be aggressive or intense. The Attract application should allure and inspire instead of intimidate or overwhelm.
- Balance the needs of experienced Surface users with those of the novice. The Attract application must be interesting enough in its own right to support repeated interaction and visitation by experienced users, but it also must be enticing enough to invite immediate, natural interaction from novices and casual observers. Beauty will attract a novice, but interest will retain an expert.
- The Attract application should always have some movement or a sense of "life." Notice that in the default Attract application a drop seems to hit the surface of the water two or three times a

minute, even when no interaction is taking place.

- It is easy to fall into the trap of creating something with too much animation, entertainment, and messaging that ends up looking like an advertisement. Users will ignore this and treat it as something passive to watch instead of something active with which to interact.
- Be aware that Surface will be unfamiliar to many of your users. Remember, Surface will become very familiar to project team members, but for many (if not all) end users the Attract application will be their first touch, first learning opportunity, and first impression.
- Ensure that the general flow of your Attract application makes sense. Users need to go from the Attract application to Launcher first before selecting and running a different application. Do not design the Attract application by assuming that users will go from it directly to another application.

Customizing the existing Attract application

The Water Attract application can be customized by replacing the background image with a custom image, by replacing the reflection image, or by adjusting the lighting effects. The background image is the most visible as it represents the bottom most layer on which the water sits, like the ground or earth, and it is what you would see when looking through the water. The reflection image is barely visible as it represents a reflection of whatever is above the water and being reflected on the surface of the water. The lighting effects represent light sources from either above or below the water (e.g., like the sun above a lake or lights inside a swimming pool).

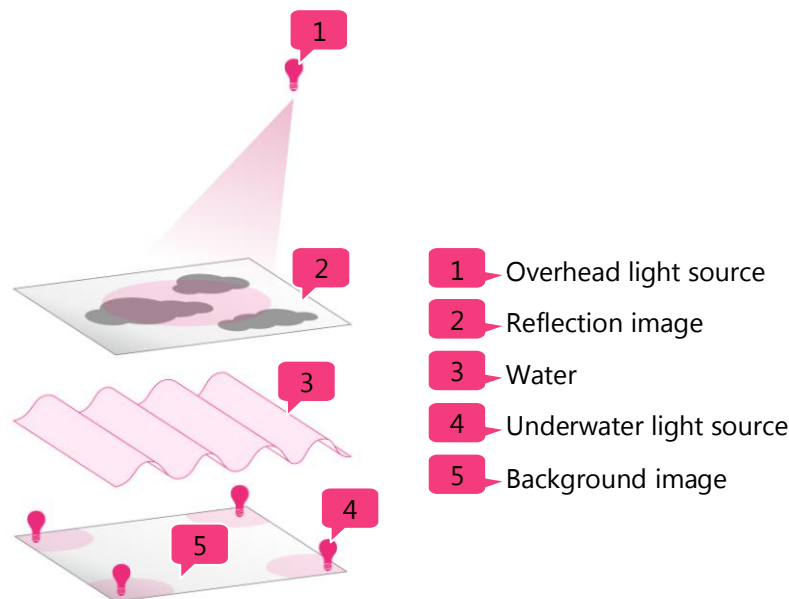


Figure 8.6: Diagram showing the different components of the default water Attract application.

A “Water Configuration Tool” is provided with the Surface SDK and Surface Runtime. Within that tool is a help file offering step-by-step instructions. Below, however, are some design considerations and recommendations.

Replacing the background image

- Use a background image that maintains the illusion of reality. The water simulation is physically quite realistic. Users may not believe unrealistic backgrounds. If photographing visual elements that will appear to be under the water simulation, do not actually have the elements underwater; this will create another layer of highlights that would be duplicative when the interactive water layer is composited over the background. Instead, lightly moisten the objects or shoot them dry-for-wet.
- Match all lighting. The Attract application has its own lighting parameters that can be customized (see below); photograph or prepare visual assets so that their highlights, if any, match the color and position of the Attract application's own built-in virtual lights.
- Design your background image so that the areas around the Access Points and the nearby image areas are clearly contrasted. It is important to help users discover and use the Access Points, which are superimposed on each of the four corners. Access Points remain findable and discoverable by virtue of their position in the corners and their contrast in color and lack of robust animation, lending them a feeling of solidity and constant presence.
- Avoid adding visuals that resemble UI controls. Providing anything that resembles UI controls or looks like explicit touch targets may affect the discoverability of the Access Points. Be careful to avoid images that look like visual buttons, interaction targets, or pieces of content. Such items can potentially result in user confusion when the action they expect to occur never does. Creating "ambient" visuals is a good goal when creating the new background image for the Attract application.
- Create a subtle embossed logo or branding on an element of the background image, rather than making the entire background the company logo and name. Remember that branding the Surface experience is broader than visually integrating a corporate mark or color scheme (see [Visual Branding Principles](#)).
- Use midtone and highly saturated colors to help make the water feel crisp. Washed out colors give the water a murky appearance and should be avoided.

Replacing the reflection image

The Attract application contains a reflection image in a layer that sits on the water's surface:

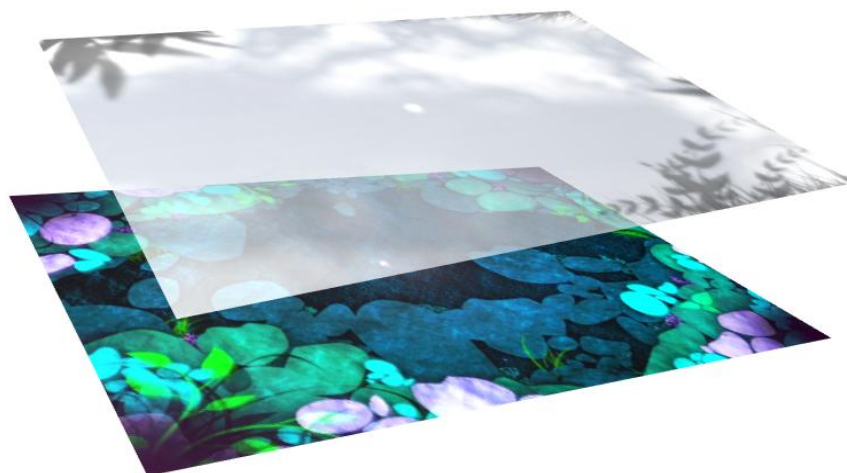


Figure 8.2: Reflection image sits on top of the water, whereas the background image sits below the water.

The default Attract application uses clouds in its reflection image, as if the water's surface were reflecting the sky. The reflection image is distorted by the water ripple effect, making the ripple itself more visible and realistic. This reflection image can be turned on or off. Additionally, its opacity can be specified. In creating a custom reflection image consider the following:

- You can incorporate multiple reflection images and rotate between them.

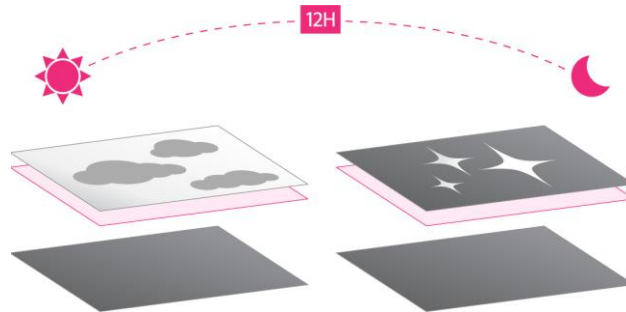


Figure 8.3: Separate day and night reflection images could switch at a specific time. Similar themes that involve subtle changes over time can build a lot of interest without being overt or garish.

- Select reflection images that fit with the background image (contextually, following the metonym that the image represents) and/or the venue's immediate physical environment (to introduce intriguing *trompe l'oeil* effects). For example, a rural background image matches nicely with clouds and trees but not as well with urban objects (e.g., buildings).
- Do not let the reflection image overpower the content that appears to be below the water. Ideally, the background image contains more information and detail, whereas the reflection image is simpler so as not to compete or confuse. For example, a bottom-up image of a tree with many branches and leaves may simply be too much detail and information for a reflection image.

Adjusting the lighting effects

The default Attract application offers two light sources. An overhead light source reflects off the water's surface and its specular light effects are calculated based on water surface movement. An underwater light source refracts beneath the water and ripples distort it along with the background image. Only one light source can be selected at a time, but it can change over time: once again, imagine time-of-day lighting in which a light source from above is used during the day and a light source from below is used at night.

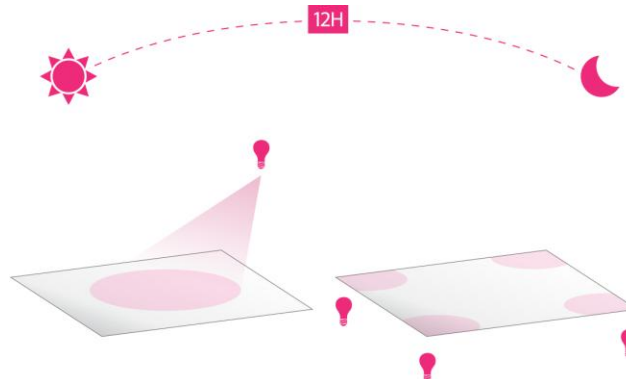


Figure 8.4: Diagram showing a change in light source over time of day.

In adjusting the lighting effects consider the following:

- Do not allow any lighting effects to visually de-prioritize or obscure the Access Points in the corners of the screen.
- Avoid harsh lighting. Create an environment that is welcoming, organic, and friendly.
- When choosing to have the light source come from beneath, it will emerge from the four screen corners, therefore, carefully consider how this will affect your background image.
- To create a softer visual effect, you can enable a soft highlight effect, which saturates the color of screen images and creates soft specular highlights.

8.1.2 Launcher

Launcher is the central hub of the Surface experience. It lets users select an application, but has many subtle aspects to its design that helps users determine not just what an application does, but how it works. In this way, Launcher acts as both an application launcher and as a critical user education tool.

While Launcher itself cannot be customized, its application-specific content can be. There are four pieces of content associated with each application in Launcher: icons, titles, descriptions, and preview pane content.



Figure 8.5: Launcher is where applications are listed, described, and accessed.

Some design considerations and guidelines are provided for each of these components.

Icons and Titles

- Icons and titles should be clear, direct, and unambiguous.
- Icon specifications:
 - Dimensions: 250 x 250 pixels
 - Color mode: RGB
 - Resolution: 96 dpi

- Bit depth: 32
- File type: PNG with transparency background
- A drop shadow should be applied from directly above and a little to the left of the icon. Drop shadow settings:
- Blend mode: Multiply
 - Opacity: 55%
 - Angle: 120 (use Global Light)
 - Distance: 10 px
 - Spread: 0%
 - Size: 5 px
 - Quality: use default settings
- Elements in the icons should be simplified, but should look photorealistic.
- Specular highlights can be applied to the objects, but they shouldn't overshadow the content itself.
- Because the content is the interface the icons should represent the archetypical content of the application or the tasks it can perform. As with other aspects of the Surface experience, an application's icon and title should be based on either the content it can manipulate or the tasks it can perform. For example, the Photos application has an eponymous title and an icon of a stack of snapshots, instantly and naturally inferring its function.
- Sometimes an application has functionality that is too complex or multi-tiered to be directly tied to one type of content; in this case a metonym or other abstraction can be useful. For example, a Concierge application might offer the services of a concierge without actually being a real or virtual agent, but users can infer its benefits from its name alone.



Figure 8.6: This is an example of the final touches being put on an icon for Launcher. From left to right: Base icon design, then with highlights, then with final drop shadow.

Descriptions

Application descriptions appear when an application is in the center of Launcher.

- These descriptions should be short, and should focus on what the user can accomplish in the application.
- They should not include any explicit instructions for the application, since the application's

interaction design should be natural enough to use without significant guidance.

- Users should be able to read the full description in under 7 seconds.
- Help orient the user towards the appropriate gestures before even opening the application by using action verbs in the description. For example, an application called “Video Editor” might have a description that reads, “Grab, drag, and re-order footage to create your own video.” This description not only illustrates what the application can do, its language (“grab”, “drag”) also infers what gestures should be used in order to interact with content.

Preview Pane Content

Application preview pane content appears with descriptions when an application is in the center of Launcher. They literally show the user how the application might be used, as if he or she was watching a live session of the application. Preview panes can show a movie clip (.wmv media file) or a slide show of sequential images, and both can include sound.

The primary benefit of preview panes is to give the user a literal preview of the application at work, as if they were watching someone else use it. This provides several critical benefits, which you should be mindful of when creating content for your application’s preview pane.

- It reinforces latent learning and is a convenient way to lead by example. Only a few images, or a single moment of video, can illustrate the appropriate gestures needed to interact with the application, removing most of a user’s learning curve. For example, the preview for the Photos application can show an animation or movie of two hands stretching or shrinking a photo and fingers flicking a few photos.
- The application preview pane need not contain an actual video. A slide show or montage of image sequences can often convey the same information as effectively as a video can, as long as the image sequence still manages to “tell the story” of what gestures yield what results in the application.
- Full-motion video this early into a user’s interaction with Surface—just a few touches away from the Attract application—is a pleasantly surprising, media-rich experience that pleases and informs the user.

There are some general techniques and guidelines to follow when creating application preview pane movies.

- The final video (or images) for an application preview pane should be 320 by 240 pixels in size.
- Only .wmv (Microsoft Windows Media® Video) files can be used in application preview panes. They should be compressed using a tool like [Windows Media Encoder](#) targeting...
 - a bit rate of 2,000 kbps (2 mbps)
 - a frame rate of 29.97 frames per second (fps)
 - a final output size of 320 x 240 pixels
- For best results, use constant bit-rate (CBR), two-pass encoding. (Hint: Windows Media Encoder’s “DVD Quality Video” preset follows most of these values.)
- Application preview panes do not provide sound or soundtracks of their own. Any audio accompaniment must also be created by you.

- [Creating Preview Movies for Surface Applications](http://community.surface.com/blogs/design_process/archive/2008/06/18/creating-preview-movies-for-surface-applications.aspx)⁹ is a step-by-step set of instructions available to help you with the technical aspects of creating a preview movie.

If an application development team doesn't have time or resources to make a full motion video they can provide images to the Surface shell and a slideshow will be created in its place.

- The slideshow consists of between 1-5 png files, which must all be exactly 640 x 480 or they will be rejected by the system.
- The slideshow uses a typical Ken Burns effect of displaying cross fades and constant movement. Since the image is bigger than the preview pane only 1/4 of the image will be shown and the center of the image will be shown more often than the edges. Take this into account when creating images.

8.1.3 Notifications

Notifications are a vital part of the Surface experience. They are needed for when the unseen, unexpected or unfortunate happens, and this makes their design and messaging critical to reinforcing a positive computing experience that is aware, responsive, friendly, and engaging.



Figure 8.7: Example screen with a Notification superimposed at one edge of the screen.

What to notify

- A Notification should inform users of events that are unrelated to the current user activity.
- The information in a Notification should be useful and relevant, but never critical. Consequently, Notifications do not require immediate user action and users can freely ignore them.
- Do not use Notifications to communicate an error in an application that is open and in the

⁹ http://community.surface.com/blogs/design_process/archive/2008/06/18/creating-preview-movies-for-surface-applications.aspx

foreground. There will most likely be a better way to communicate the error in the context of the application.

- When an application crashes the system will give the error message so there's no need for that application to send a notification.

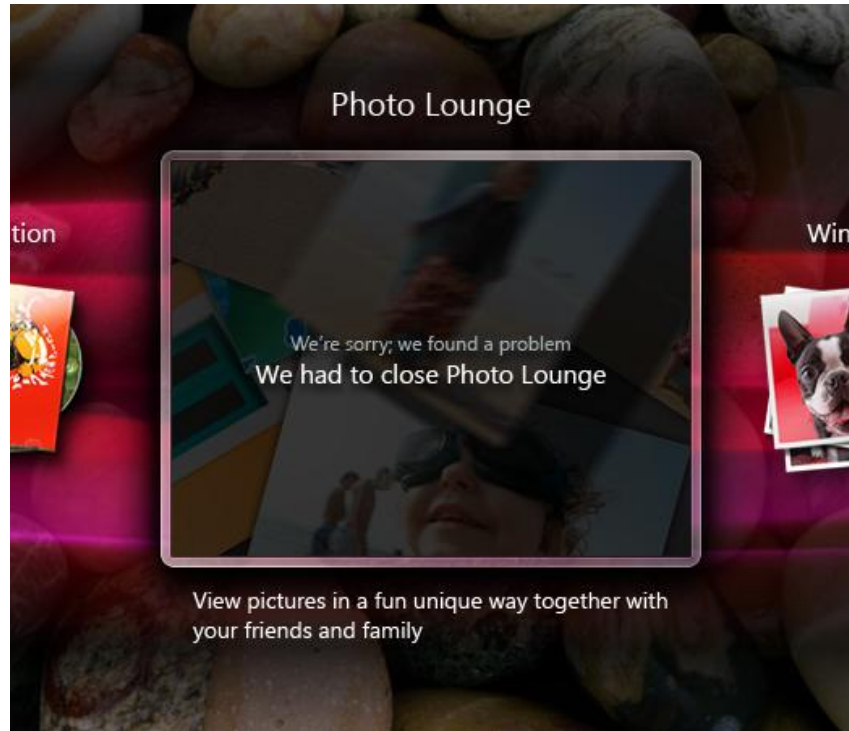


Figure 8.8: This error message isn't hosted by the Photo Lounge application, but Launcher, both offering an explanation and putting the user one step closer to doing something new (i.e., choosing a new application).

- Notifications should be used for asynchronous information only. If the information is the immediate, direct result of users' interaction with your application, then display this synchronous information directly within your application using in place UI.
- Notifications are not suitable for rapidly changing, continuous, real-time information. Examples include processing progress, stock quotes, and sports scores. Once again, for such information use the application's own UI.
- Display critical information requiring user action directly within your application using an interface that demands attention and cannot be easily ignored.
- Use an alternative feedback mechanism such as log file entries or e-mail for messages targeted at IT professionals. IT professionals strongly prefer log files for non-critical information. Furthermore, servers are often managed remotely and typically run without any users logged on, making Notifications ineffective.
- Do not notify of successful operations, except in the following circumstances:
 - Security. Users consider security operations to be of the highest importance, so notify users of successful security operations.
 - Recent failure. Users do not take successful operations for granted if they were failing

immediately before, so notify users of success when the operation was recently failing.

- Prevent inconvenience. Only report successful operations when doing so might avoid inconveniencing users. Consequently, notify users when a successful operation is performed in an unexpected way, such as when an operation is unexpectedly lengthy or completes earlier or later than expected.

When to notify

Ideally, users immersed in their work will not see your Notifications at all. Rather, they will see your Notifications only when their flow is already broken. In *Flow - The Psychology of Optimal Experience* (1991, Harper Perennial), Mihaly Csikszentmihalyi says that users enter a flow state when they are fully absorbed in activity during which they lose their sense of time and have feelings of great satisfaction. Effective Notifications help users maintain their flow by presenting useful, relevant information that can easily be ignored. The Notifications are presented in a low-key, peripheral way, and they do not require interaction. Do not assume that if Notifications are modeless they cannot be an annoying interruption. Notifications do not demand users' attention, but they certainly request it. You can break the user's flow and therefore should avoid doing the following:

- Displaying Notifications that users do not care about.
- Displaying a Notification too often.
- Using several Notifications when a single Notification is sufficient.
- Using sound when displaying a Notification.

Display a Notification based on its design pattern:

Pattern	When to notify	How often to notify
Action success	Upon completion of an asynchronous task. Notify users of successful actions only if they are likely to be waiting for completion, or after recent failures.	Once
Action failure	Upon failure of an asynchronous task.	Once
Non-critical system event	When an event occurs and the user is active, or the condition continues to exist. If this results from a problem, remove the currently displayed Notification immediately once the problem is resolved.	Once when the event first occurs Redisplay once every ten minutes if users must resolve within an hour, or once every hour if users must resolve within a day
Optional user task	When the need to perform a task is determined and the user is active.	For security-related tasks, once an hour For all other tasks, once a day for a maximum of three times
FYI	When the triggering event occurs.	Once
Feature advertisement	When the user first performs an action where the feature is relevant.	Once a day whenever the feature is relevant, for a maximum of three times

How to notify

- The duration for a Notification can be customized. By default leave the Notification on screen for 9 seconds. If modifying the time duration, use the minimum time possible while still giving users enough time to read the Notification.
- Notifications should never be technical in nature. A user should not have their Surface experience interrupted by messages indicating specific failures of subroutines, exceptions, and errors.
- Keep all Notification messaging short, friendly, and helpful. Apologize for inconveniences (if needed) and plainly describe what happened.
- Give users clear and specific next steps, especially if a triggered action has no immediate visual effect, such as printing. If an application or Surface itself needs to take action, notify the user as to what is about to take place, should he or she need to wait.
- Give the Notification its own graphical shape, background, or other visual device so that its typography has sufficient contrast against the rest of the on-screen elements.
- Notifications must be read from one primary direction, so use large font sizes if the message needs to be scanned from multiple angles by more than one possible user. Secondary or contextual information in the notice can be rendered in smaller type for the benefit of a single user if desired.

Design example



Figure 8.9: This notification is a positive one, and necessary if the printer isn't in close proximity to Surface. Note its shaded background as a holding element and for improved legibility.

In this example the user has ordered photos to be printed at the front desk of his or her hotel. While in the Concierge application, the Photos application notifies the user that his or her photos are finished. This Notification can be ignored. Alternatively, users can choose to quickly navigate back to the Photos application by touching the Photo's icon.

8.1.4 Out of Order Screen

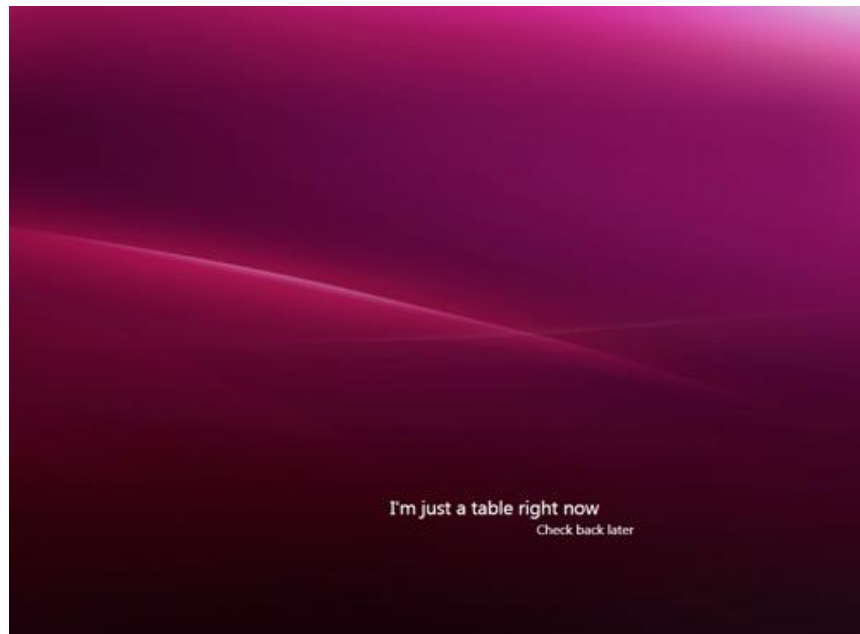


Figure 8.10: The Microsoft Surface Out of Order Screen is beautiful in its own right, as well as friendly and light-hearted.

The background images and text messages of the Out of Order screen are customizable. This screen appears when Surface is out of order.

Consider the following if you are replacing the images and text:

- Keep the message friendly, open, and interesting. Just because Surface is not available at the moment does not mean it has suddenly lost its engaging personality. Avoid all technical language.
- Avoid promises as to when Surface will be available again.
- Avoid specific imagery on the Out of Order screen, and instead use richly textured backgrounds. This ensures that no user casually tries to interact with something on-screen that is not interactive.

8.1.5 Application Backgrounds

A strong contrast between an application's foreground elements (interactive elements such as content and controls) and its background (the zone in which touches traditionally trigger no events) is vitally

important to the clarity and simplicity of a natural interaction model. Without instructional text or traditional GUI/WIMP controls, it is this separation of foreground and background that instructs the user as to what events their touches might trigger.

There are several considerations in the design of an application background.

- While backgrounds do not trigger events when touched, they should nevertheless respond to a user's touch, so that the user knows his or her touch has been successfully detected by Surface. This can be accomplished through animations and motion graphics that respond to touch globally across the entire application, not just within controls such as buttons.
- Backgrounds should be attention-neutral. They should truly recede into the background of the user's perception, in order to subtly indicate that touches will probably not result in an action. This also allows any animated touch responses to stand out and be easily visible.
- Dim, subdued tones and muted, desaturated colors are often the best choices for application backgrounds. This is especially important when content is highly variable (dynamic or user-provided), as foreground colors will be arbitrary or unpredictable. In addition, the Surface screen is fairly large, so very bright or colorful backgrounds will be quite fatiguing for most users.
- Just because a background is attention-neutral does not mean that it must lack character or beauty. Restraint is the key. If the background is understated and the foreground elements are nicely separated from it, then it should suffice.

Design Examples

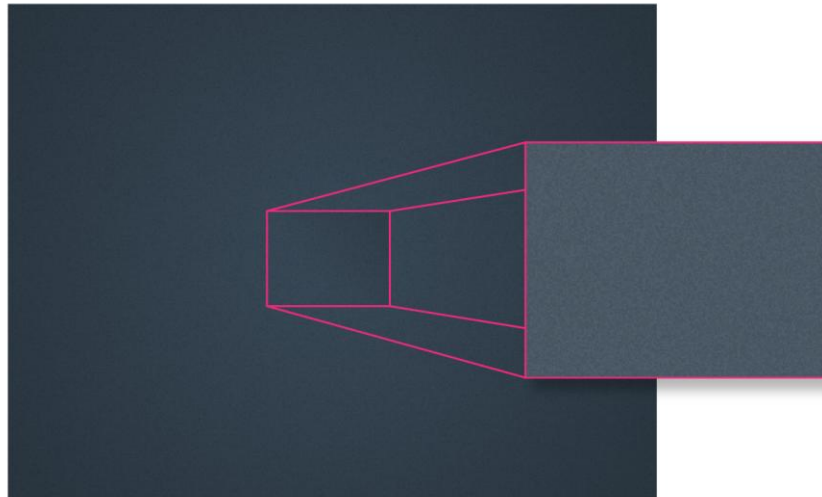


Figure 8.11: The feeling of an application background should be that of a dark stage helping the content take center stage.

Application backgrounds should support foreground elements, but should not feel like one itself. The tonal range of the background should be 50% grey or darker. Default backgrounds will use the felt metaphor. This effect is obtained adding texture via the "Add Noise" filter in Adobe® Photoshop®.

8.1.6 User Assistance / Help

Ideally, a Surface experience should not require a Help system. Gestures should be natural, actions should be clear from the proper design of user affordances on controls and content, and goals should naturally

come from the user (albeit assisted by Launcher's guidance for each application.)

Even the most natural user interfaces may sometimes need to give users a helping hand. Like with so many other aspects of the human-computer interface, Surface has its own way of guiding users through unfamiliar applications.

Latent Learning: Showing By Example

One way to assist the user is by employing latent learning to proactively illustrate or suggest how a control should be used (see [Latent Learning: Transitions as Affordances](#)). "Latent learning" describes how learning occurs passively by repeated observation. For Surface experiences, this can be done on multiple levels to suggest appropriate actions and gestures to the user before they need them.

- Application preview videos, as small and as brief as they are, can effectively illustrate both high-level goals within an application and the gestures needed to accomplish them (see [Launcher](#)).
- Ensure seamless transitions to provide clues about state and control usage (see [Latent Learning: Transitions as Affordances](#)). For example, showing content within a ScatterView control "spill" across the screen underscores that each content element is discrete and can be moved, thrown, and rotated. Showing a list box's contents scroll into place illustrates that the control holds more content than can be shown on-screen, and that a flick or drag is probably all that is needed to access that content.

Passive (System-Delivered) Assistance

Microsoft Surface can offer help content in a passive way, not requested by the user and not explicitly tied to animations, transitions of content, or controls. This kind of help content exists within an application, but is more "ambient" and more direct than latent learning techniques, while also being less didactic and broader than help specifically shown upon the user's own request.

- Help can be delivered as part of a brief introductory animation or demonstration sequence immediately after application loading. For example, novel or unclear gestures can be demonstrated briefly as soon as the application launches. If an application needs to wait for a user to input data, the background of the application can give visual clues or hints, or even play back past usage sessions. This can be especially useful for games.
- Sometimes brief instructional text is required to initiate an experience, but it should be short and easily readable by all users, and utilize supporting visuals to clarify the next steps. This can be the case when object recognition and interaction are key parts of a Surface experience. For example, text prompts may be needed to help users understand that nearby objects can be placed on the Surface screen. However, these text prompts can be kept concise if they appear within a visual silhouette that matches the "footprint" of the object to be placed on the Surface screen. In this way, short written instruction can be sited within content or controls that can be moved and freely oriented.
- Of course, UI controls are labeled with words or universally understood symbols. These labels are a form of passive assistance. See the next section, [User Interface Language](#), for more information.
- User feedback does not have to be limited to just visual and/or audio tones. When appropriate, voice feedback can be a complementary form of audio feedback. Voice can also be another delivery channel for assistance or instruction. In a game, for example, instead of just showing the text "Start", complementary voice audio could also be played saying "start", which happens to reinforce a 360 UI as audio is not oriented to any one side of the Surface screen. Do not, however, rely on audio as the only form of feedback or instruction as ambient noise and/or low Surface

volume settings may make it difficult for users to hear.

Active (User-Requested) Assistance

There are times when a user actively requests help. Microsoft Surface experiences should provide these if needed, but in a different and more NUI-appropriate form than is typically seen on computers and other GUI/WIMP computing experiences.

- Help should lead by example whenever possible. It should be direct, but not didactic, illustrating as it instructs. Animated or video content can be easier and faster to visually digest than written Help content. Usually it is only necessary to show what gesture is needed to yield a certain event or result. The user should still be encouraged to explore without feeling intimidated about poor results or mistakes.
- Help should require as little reading as possible. Help facilities should educate every user from all sides of the screen. Large amounts of text will only allow reading by users on one side. Help should, instead, be visual and directly instructive, illustrating gestures, actions, and control usage in a way that is contextually appropriate.
- Demonstrating gestural input *in situ*—that is, right on the screen itself, interacting with actual content—is another powerful way of delivering help content. For example, this could take the form of realistic hands appearing on the screen, auras, or other abstract representations of hands or objects over the on-screen content. Such animated demonstrative sequences should be accessible from a recognizable button, icon, or content element. The demonstrative sequence can vary with the application’s state or the user context. Such visualizations should be easily dismissed if not needed, and should not slow down the overall pace of the experience.
- Help can appear as a piece of content. For example, if a user needs help using a ScatterView control, one ScatterViewItem can contain user-assistance video content, and be visually identifiable as a different content type that can offer assistance. This maintains the metonym used and can be helpful without completely breaking down the user’s suspension of disbelief.

In such cases where users actively request help, the design of the request mechanism itself is a challenge. The solution depends on the Help content and pairing it as close as possible to the part of the application to which it pertains. Global Help content should be accessible via a control located in an application-neutral location, whereas local Help about a control or object should be accessible from the object itself (e.g., context menu).

8.2 User Interface Language

Because most Surface experiences will be in a commercial environment, the time that users remain using one application is limited, and the time it takes to assist users must be as fast as possible.

Use text or voice audio to help users, but minimize this type of help because users learn best by observing and then interacting. Observation and interaction are simpler methods than text to convey information, but it can clarify or more easily explain difficult concepts. Although voice audio is not used very often, it can quickly and easily inform users in ways that text cannot. However, text and voice audio can draw people away from the Surface content and experience, taking up most of their limited time in a commercial environment.

When using text or voice audio, incorporate them in a way that complements the user experience, rather than contradicting it. Surface does not speak like a computer. Many users view Surface as a magical

device that belongs in science fiction films. Text and voice audio need to satisfy any lofty expectations.

The following examples demonstrate where to add text or voice audio to help Surface users:

- Clarify the purpose of buttons, such as "I'm done" or "Remove my photos."
- Indicate when an action has been completed, such as "This song has been added to the playlist."
- Explain that an action is occurring, such as "We're closing everything and starting a new experience."
- Describe a product, location, interaction, or feature of the application.

A New Language for a New Product

These guidelines describe how to implement and design text in Surface applications, and then describe the following principles which apply to all Surface text and voice audio:

- Casual and Comfortable
- Clear and Concise
- Direct and Personal
- Not Computer-Based

Note: Although all the Surface interface language guidelines provided in this section apply most directly to text, the 4 principles listed above should be followed for both text and voice audio.

In the following examples, text inside brackets, for example, [I'm done], represents interface buttons.

Implementation and Design

The following guidelines relate to implementing and designing text. These guidelines might apply to an interaction designer or an application developer.

Enough Time to Read

Text should appear long enough for users to read the message before the text disappears from the screen.

Most people can read five to six words per second. For every five words, keep the message visible for one second. For example, display "We found a problem; we will now close all the activities and content" (13 words) for at least three seconds. Also, consider how long users need to focus on your message and become acclimated to it. This might take an additional one or two seconds.

This guideline applies to any messages that are in notifications, errors, status information, or other text that appears for only a short time.

Text on a Button

Use text on a button to explain the purpose of the button to the user beyond a graphical explanation. Make sure the language is clear, concise, casual, and comfortable. Avoid computer-based commands that you would normally find on traditional computer interfaces. For more information, see the principles below.

Use text when users expect to see it, such as with a checkbox, a list of items, or a radio button.

For difficult-to-explain commands, use graphics and text on the button. As a general guideline, use text as little as possible, but it is better to use text than to allow usability issues.

Titling Content

Add titles to content that contain information. For example, place a title at the top of an album that lists songs, or place a title at the top of a card that contains a description of a product. However, it is not necessary to place a title on a photo, video, or album cover. The first frame of the video could contain the title.

Titles of content are the only text where the first letter of each word is capitalized. In all other scenarios, such as buttons, descriptions, and out of order text, only capitalize the letter of the first word. See the *Casual and Comfortable* principle below for more guidelines on capitalization.

Casual and Comfortable

Casual language is familiar, informal, conversational, natural, colloquial, and possibly slightly idiomatic. However, *Casual and Comfortable* text cannot be too colloquial or idiomatic. Otherwise, it is no longer *Clear and Concise*, and it becomes costly to localize.

Note: Although there are no current localization plans for Surface, keep localization in mind to avoid unnecessary difficulties and complications in the future.

Comfortable language is easily accepted and evokes positive emotions. It is organic, calming, and decreases tension that the user might be feeling. Comfortable language tells the user that Surface is easy to use.

Casual and Comfortable language should adhere to the following guidelines:

If the language is too casual, it will not be comfortable

- If the communication is too casual, it may not be clear.
- Do not use slang, colloquial, and idiomatic phrases that might be considered silly or are not widely used. Carefully select casual words.

Correct: [I'm done]

Incorrect: [I'm outta here]

Correct: The playlist is full

Incorrect: The playlist is maxed out

Use informal language

- Write informally, as if speaking to a friend.

Correct: I'm just a table right now
Check back later

Incorrect: Out of order – The table has experienced a minor malfunction

Correct: We can't open Photos

Incorrect: The Photos application has failed to launch

Avoid using punctuation

- When writing interface text, avoid using punctuation whenever possible.
- Only use punctuation when writing multiple sentences, using a hyphen, using a dash for emphasis (em dash,) or using an apostrophe for a contraction.
 - Correct: Start a new experience
 - Incorrect: Start a new experience!

 - Correct: Add songs from the albums to the playlist
 - Incorrect: Add songs from the albums to the playlist.
- The following example shows when it is okay to use punctuation for a description with multiple sentences.
 - Correct: Plan A includes 5,000 hours with 400 Anywhere Minutes. Calls are free on weekends and after 7PM on weekday nights. Calls to your Favorite Friends are also free.
- The following example shows when it is okay to use a hyphen.
 - Correct: Enjoying photos and videos are hands-on activities
- Use an em dash instead of an en dash to add emphasis.
 - Correct: Photos—Explore your memories
- The following examples show how to use apostrophes for contractions.
 - Correct: We're closing everything and starting a new experience
I'm not done yet

Avoid overusing capitalizations

- Use capitalization with brand names, like Microsoft Surface.
- Start each sentence with a capital letter.
- Do not start every word with a capital letter.
 - Correct: [Continue my activity]
 - Incorrect: [Continue My Activity]

 - Correct: Remove a song to add a new one
 - Incorrect: Remove a Song to Add a New One
- Start every word with a capital letter if the text is an important title or label.
 - Correct: Search Our Choices (title of a card in Concierge)
Hotels & Motels (subcategory in Concierge)
- Do not use all capital letters because it is much harder to read.
 - Correct: [Remove my photos]

Incorrect: [DELETE FILES]

Correct: The playlist is full

Incorrect: THE PLAYLIST IS FULL

Avoid using single words as commands to the user

- In addition to being impersonal, reminding users of computer conventions, and sometimes being unclear, single-command words are not comfortable or casual. You would rarely speak that way to a friend; besides, it sounds demanding and stressful.
- *Whenever possible*, do not use a single word on a command control, like a button. Single words are acceptable when you are constrained by space in the user interface.

Correct: [Remove my music]

Incorrect: [Clear]

Correct: [I'm done]

Incorrect: [Reset]

Do not apologize

- It is hard to maintain a valuable brand and a comfortable experience if Surface text blames itself and makes Surface look like an unreliable system.
- Do not apologize.

Correct: I'm just a table right now

Incorrect: I'm sorry for the inconvenience

Correct: This video can't play

Incorrect: I'm sorry, but we can't play this video

- Instead of using the word "sorry," use "unfortunately."

Correct: Unfortunately, we found a problem

Incorrect: I'm sorry; we found a problem.

Correct: Unfortunately, we can't read your memory card

Incorrect: I'm sorry, but we can't read your memory card

Clear and Concise

Clear language is easily understood.

Concise language is focused, well-crafted, and uses as few words as possible. Keep in mind that Surface is in a commercial environment where users may have very short interactions.

Combine clarity and conciseness to work together. Using more words can add to the clarity, but it makes the information less concise. Likewise, too few words make the information more concise but less clear. Satisfy both principles with all Surface text and voice audio.

Clear and Concise language should adhere to the following guidelines:

Say what the user wants

- When the text is a button or interaction, write exactly what the user wants to say, and say it the way that the user would say it.

Correct: [Remove my videos]

Incorrect: [DELETE FILES]

Correct: [Continue my activity]

Incorrect: [Go Back]

Be specific

- Do not assume that the user knows the application. More detail may be necessary.

Correct: Your songs have been removed from the playlist

Incorrect: DELETED

- Try to be more specific and still be truthful to the scenario.
- Instead of using "file," tell the user what the file is.

Correct: [View my photos]

Incorrect: [Upload all files on device.]

Correct: [Remove all the songs]

Incorrect: [Remove all the files.]

- Instead of using "application," tell the user what specific application is being referred to.

Correct: [Restart Video Puzzle]

Incorrect: [Reset the application.]

Correct: We had to close Photos

Incorrect: The application closed unexpectedly.

Use few words

- Without compromising clarity, use as few words as possible.

Correct: [Close everything]

Incorrect: [Remove my files, close the applications, and return to the Attract Mode]

Correct: This video can't play

Incorrect: Your video is corrupt or it is an unsupported file type.

Add clarity to the interaction design

- Add text to the interaction design to inform the user what to do.

Correct: Use the keyboard to start a new search
 Drag songs from the albums to the playlist

- When the user might be confused or might not understand what a gesture or action has accomplished, tell the users that they have completed an interaction.

Correct: We printed your directions
 This song has been added to the playlist

Note: Do not add clarity whenever *any* gesture or action is completed. Only add this text when the user is *prompted* to complete the gesture (like in a game) or when the user might be confused about the interaction (i.e. when standard audio and visual feedback seems insufficient).

- If user information, such as a credit card number, is kept on the application, users are exposed to computer hacks. Users want to know what is done with this information, so confirm that personal information will be removed from the system.

Correct: [Permanently remove all my personal information]
 We permanently removed all your personal information

Direct and Personal

Direct language speaks clearly to users when written in first-person, not second-person.

Personal language simulates conversation, rather than just listing commands and information. Without compromising clarity and comfort, establish a connection with users by interacting with them in a personal way. The tone should be similar to an e-mail message, not a computer interface.

Direct and Personal language should adhere to the following guidelines:

Speak from the user's perspective

- Do not speak to the users when they are making decisions. Allow the users to speak for themselves.
- When the user is choosing an option, speak from the user's perspective with "my."
Correct: [Clear my photos]
Incorrect: [Delete your photos.]
- When the user is giving information to the application, speak from the user's perspective with "I."
Correct: [I'm not done yet]
Incorrect: Are you done yet?
 [Yes] [No]

Correct: [I changed my mind]
Incorrect: [Cancel]

Speak from the Surface unit's perspective only when necessary

- Try not to speak from the Surface's perspective.
- If it is necessary to speak from the Surface's perspective, use "we."
Correct: Everything is closing to start a new experience.

Incorrect: The Surface unit is closing everything and starting a new experience

Correct: Only one memory card can be read at a time.

Incorrect: The Photos application can read only one memory card at a time.

Don't give Surface a "personality"

- Except when developing an application for children, do not turn Surface into a specific person with a personality.

Correct: [I'm ready to begin]

Incorrect: I'm glad to see you again. Touch the **start** button to begin.
[START]

Correct: [I'd like to try again]

Incorrect: Don't give up! You can do it!
[TRY AGAIN]

- When developing an application for children, associate the personality with the character or the game, not with Surface.
- If there is no application-specific character speaking, define the personality of the voice audio according to these interface language principles and guidelines.

Speak to users directly

- When the user is waiting, when an error occurs, or when the user must be given a message, speak directly to the user from the Surface's perspective.

Correct: We're removing your photos

Incorrect: All photos are being removed from Microsoft Surface.

Correct: I'm just a table right now

Incorrect: This Surface unit is temporarily out of order.

Do not ask questions

- Questions are potentially confusing, and they take time to answer.
- When users are asked questions, they must think of the answers. When users are given a button with a clear purpose, they do not have to think about anything except for that purpose.
- Instead of asking users what they want to do, provide a direct experience by assigning the user's options to the interface.

Correct: [Close everything]
[I'm not done yet]

Incorrect: Are you done?
[Yes] [No]

Correct: [Remove all my photos]

[I changed my mind]
Incorrect: Are you sure?
[Yes] [No]

Do not give information and then choices

- The average time users spend in an application in the commercial market is too short for users to read information, make a decision about what button to press, and then press that button. Giving users information and then choices violates the *Clear and Concise* principle.
- Many computer conventions are confusing to computer novices. Some users might not own computers.
- Giving users information and then choices reminds users of computer experiences. It is very rigid and inhuman, violating the *Casual and Comfortable* principle and the *Direct and Personal* principle.
- Instead of giving information and options, speak directly to the users in a concise and casual format.
- Do not ask the user a question and then give multiple options.

Correct: [I agree with the terms]
[I disagree]

Incorrect: Do you agree with the terms?
[Yes] [No]

Correct: [I'd like to continue]
[I'd rather go back]

Incorrect: What would you like to do?
[Continue] [Go back]

- Do not tell the user the status and then give multiple options.

Correct: [Remove my videos]
[I'm not done yet]

Incorrect: You are about to delete your files.
[Okay] [Cancel]

Correct: [Close my activity]
[I'm not done yet]

Incorrect: You are about to close the application.
[Okay] [Cancel]

Not Computer-Based

Avoid any language and language styles that are common to computers but not found in everyday language.

Computer-based terminology reminds users of the rigid, impersonal, and overly formal language that they can find in their computer. This language is not *Casual and Comfortable* or *Personal and Direct*.

In addition, computer-based terminology often violates the *Clear and Concise* principle and the goal for the application to be aware and responsive by causing confusion.

Not Computer-Based language should avoid specific terminology.

Avoid the following terminology

- **Application** — This term might be the best word to describe the software in general, but avoid it by referring to the name of the application or to the Surface experience in general.
 - Refer to the specific application.
 - Correct: [Close Photos]
 - Incorrect: [Exit the application.]
 - Refer to the entire Surface experience.
 - Correct: [Start a new experience]
 - Incorrect: [Close all applications.]
- **Attach** — In everyday language, users do not say “attach this paper to the envelope you’re sending.” They say “could you put this paper in the envelope?” Instead of “attach,” use “add,” “include,” or a longer phrase like “put this in” or “stick this on.”
 - Correct: [Include my photos]
 - Incorrect: [Attach all files.]

 - Correct: [Add a photo]
 - Incorrect: [Attach a photo.]
- **Cancel** — In everyday language, users do not say “cancel what you’re doing.” They say “could you stop for a moment and come here, please?” Instead of using “cancel,” speak directly to the purpose.
 - Correct: [Remove my photos]
 - [I’m not done with my photos]
 - Incorrect: Delete all files.
 - [OK] [Cancel]

 - Correct: [I changed my mind]
 - Incorrect: [Cancel]
- **Confirm** — In everyday language, users do not say “confirm” when they need to confirm information. They would just ask for clarification. Instead of using “confirm,” speak directly to the purpose.
 - Correct: [Remove my music]
 - [I’m not done yet]
 - Incorrect: Your files will be deleted.
 - [Confirm] [Cancel]
- **Delete** — In everyday language, users do not say “delete your plates after you’re done eating.”

They say “clean your plates” or “remove your plates.” Instead of “delete,” use “remove.”

Correct: Remove my videos

Incorrect: Delete my files.

- **Exit** — In everyday language, users do not say “exit the room when you’re done.” They say “leave the room” or “go to the garage when you’re done.” When you are giving a control command, use “I’m done” instead.

Correct: [I’m done]

Incorrect: [EXIT]

- When the term is in a sentence, use “close” instead.

Correct: [Close Concierge]

Incorrect: [Exit the application.]

When the term is used to tell users to leave, use “go” if possible.

Correct: Please go back to the Surface menu

Incorrect: Exit to the Application Launcher.

- **Failure** — In everyday language, users do not say “I just detected a failure in the car.” They say “there’s a problem with the car.” Instead of using “failure,” use words like “problem.”

Correct: Unfortunately, we found a problem

Incorrect: Application Failure Detected

- Instead of using “failed,” use words like “can’t” or “wasn’t able.”

Correct: Photos wasn’t able to open

Incorrect: Application Failed to Load

- **File** — In everyday language, users do not say “would you like to look at my files? I just got them developed.” They say “would you like to see my photos?” Refer to the specific type of media, such as photos, music, videos, postcards, or pages.

Correct: [Rearrange my photos]

Incorrect: [ORGANIZE FILES]

- When referring to multiple types of media, use “media” instead.

Correct: [Remove my media]

Incorrect: Delete all files?

[OK] [Cancel]

- **Load** — In everyday language, users do not say “load your food into the bowl.” They say something casual like “serve yourself.” Instead of using “load,” use terms like “collect,” “add,” “open,” “search,” or “find.”

Correct: We are collecting your music

Incorrect: Loading all files...

Correct: We're opening your profile

Incorrect: We're loading your profile.

- **OK** — In everyday language, users do not say "I need you to go to the store today: OK or cancel?" They say "could you go to the store for me?" Instead of using "OK," speak directly to the purpose.

Correct: [Remove my information]
[I'm not done yet]

Incorrect: We are about to remove all your information.
[OK] [Cancel]

Note: "Okay" is fine to use in a sentence if it aligns with the other interface language principles.

- **Reboot** — In everyday language, users do not say "I need to reboot the microwave." They say "I need to restart the microwave." Instead of using "reboot" to close applications, use "close."

Correct: [Close everything]

Incorrect: [Reboot the table.]

- Instead of using "reboot" to restart applications, use "restart."

Correct: [Restart my game]

Incorrect: Would you like to reboot the game?
[Yes] [No]

- **Session** — In everyday language, users do not say "are you ending your basketball session?" They say "are you done playing basketball?" When the term is a command, use "done" instead of "session."

Correct: [I'm done]

Incorrect: [End Session]

- When the term is in a sentence, use "experience" instead of "session."

Correct: [Start a new experience]

Incorrect: [Start a new session.]

8.3 User Interface Controls

The following section describes several Surface UI Controls which are included in the Surface SDK or in our own applications. Each of these controls represents a fundamental unit of interaction. These units can be brought together to compose an overall interaction scheme.

As you read this section, it is important to remember the principle of direct interaction in building Surface experiences. In keeping with this principle, designers are generally discouraged from placing traditional UI controls within their applications. As you read about each of the controls, try to think of them as fundamental interaction units, rather than as a particular visual element or metaphor: a `SurfaceButton`, for example, is not, at its most basic level, a button at all. In truth, it is a region of the screen a user touches to complete an immediate action. To create Surface experiences, you will need to reconsider two fundamental ideas in decreasing order of importance:

1. The select/complete cycle

2. The metaphor used to represent the control

Traditional GUI experiences are often composed with selection/completion cycle: the user selects the desired content or parameters, then presses a button to complete a command. This cycle has its place, but is generally non-preferred in Surface applications. A Surface experience should be direct and immediate, so that actions complete as options are selected. This may be challenging, but, when completed correctly, will result in more compelling experiences.

If, after reconsidering the select/complete cycle, the designer determines that a SurfaceButton is still necessary, the next step is to reconsider the underlying metaphor employed in a button. The metaphor of a mechanical device, a button, invoking an action is a useful one, in that it maps to a real-world interaction, but it is not the only viable one. As you build experiences for Surface, be careful to not limit your thinking to the mechanical metaphors traditionally mapped to UI controls.

The following sections explain the UI controls we offer to help you in building your applications.

8.3.1 Alternative Designs

For each control, we have included a description of the interaction unit it provides. We then describe the elements of its behavior and look and feel which can be easily modified using the Surface SDK. For several controls, we also provide alternative design patterns which avoid the use of that control, in order to create a more Surface-like experience. In some cases, the design pattern we describe will not cover the full range of situations in which the fundamental unit of interaction might be required. When confronted with such a limitation, your first instinct should be to extend these design patterns, or to otherwise build interaction schemes that are in keeping with the Surface experience we have defined. The use of traditional UI controls should generally be thought of as a lesser alternative.

8.3.2 Redesigning Existing Controls

The controls which are part of the Surface SDK have been extensively researched and built for maximum flexibility allowing you to apply your own custom look and feel. A visual look and feel is provided as a default, but can be easily replaced with your own schemes, while maintaining the underlying behavior of the control. WPF, the toolkit in which the SDK is built, provides a great deal of flexibility in defining the visual look & feel and behavior of each control.

When redesigning controls, it is important to keep in mind some of the underlying principles and guidelines put forth in Sections 2.2 ([Interaction Design](#)) and 2.3 ([Visual Design](#)):

- The Surface experience is predicated upon direct and natural manipulation of content, thereby enabling users to interact directly with content and therefore minimize the presence of traditional interface controls seen in the GUI or desktop computing paradigms. Visible controls should be kept to a minimum; always think about how the user could accomplish similar tasks through interacting with content directly.
- The controls included in the Surface SDK have been designed to be neutral in tone and branding; they were explicitly designed to have a minimal impact on the experience, and to pair well with almost any content or color scheme. When redesigning controls, retain the understated minimalism of the controls in favor of drawing the user's attention with actual content. Do not let the ornamentation, texture, or color of a control draw attention to itself.
- Animations, transitions, effects, and other details are absolutely encouraged for interaction feedback and communicating a control's state to the user. Save any detailing, glows, illumination,

movement, or other visual ornamentation for responding to the users' gestures. This is part of what makes the Surface experience feel aware and responsive.

- If the user will need a set of various controls to complete a task (such as a multi-step questionnaire or a sequence of disparate inputs), phase the appearance of these controls over time whenever possible. This way the experience is "scaffolded," progressively disclosing controls and complexity to the user over time, and not overwhelming the experience at any one step with control- or input-heavy screens. Be sure to use smooth, seamless transitions and animations to help maintain users' context and sense of place.

8.3.3 *Creating All-New Controls*

The advantage of visually redesigning an existing control is that you will benefit from the large amount of research and design work which has gone into its underlying behaviors. This work goes far beyond their basic look & feel, and includes a great deal of fine detail relating to their sometimes very subtle behaviors. In creating all new controls, be very careful to ensure your designs are in keeping with these sometimes subtle details.

If ultimately you decide to design your own controls, each of the following must be considered, designed, researched, tested, and implemented:

- What is the fundamental interaction unit of the control? How can that unit be abstracted for reuse?
- What are the states of the controls for each of the various touch-state combinations?
- What will be the model for how the control interacts with multiple fingers? How does that model fit with the overall model of the application that is being developed, and with the models developed by the Surface team at Microsoft and elsewhere?
- Is the new control in keeping with the Surface design principles?

Many of the previous guidelines in Sections 2.2 and 2.3 are relevant to the design of new controls.

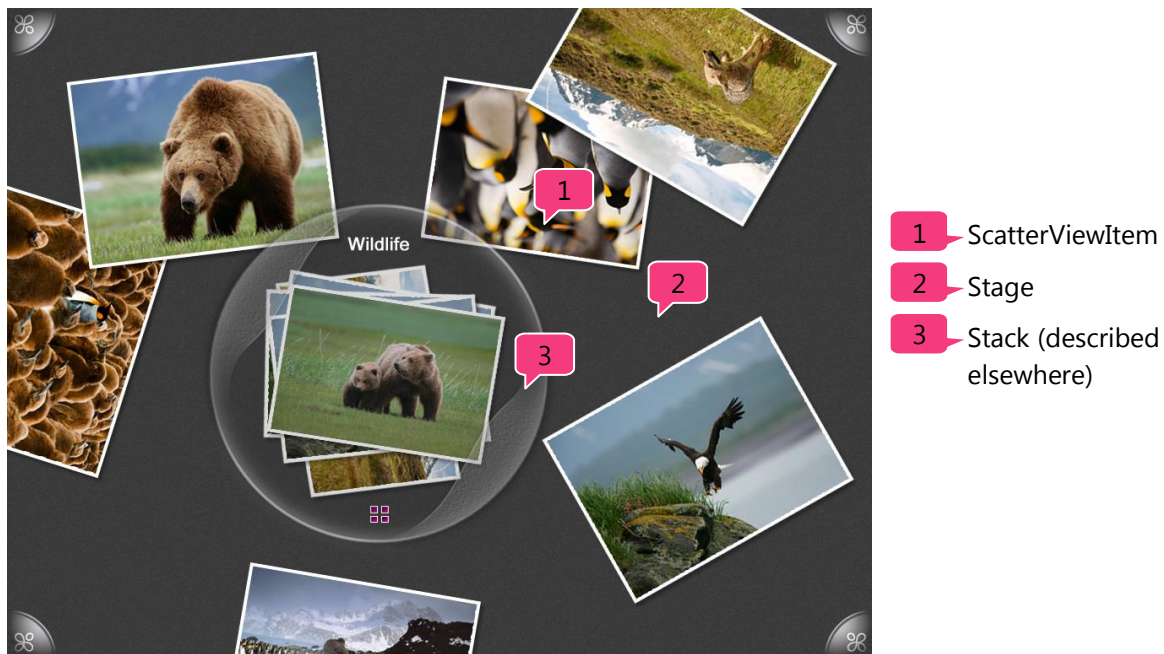
- New controls should support the display and direct manipulation of content as a substitute for more traditional GUI-style controls. Controls should be "superreal" objects on a table, holding directly interactive content, reacting with realistic physics, but exhibiting unexpected behaviors that are discoverable, repeatable, and natural.
- As with customization of existing controls, simplicity and understatement is the best approach for visual design. Controls should support content, not distract from it, so the most minimal design approach is often the best one. This also helps in making new controls flexible, re-usable, and easily customized for future projects or products.
- Regardless of what the new control does, if it looks like content and reacts sensibly to a user's touch (hopefully revealing or exposing something about itself or how it can be interacted with), the user will consider it content and no other visible control will be needed.
- Resist the temptation to simply touch-enable existing controls from related frameworks, such as Microsoft Windows Presentation Foundation (WPF). These frameworks may perfectly supports desktop GUI paradigms, but in doing so rely on large amounts of visible indirect controls, which is anathema to the ideal Surface experience. The more innovative and Surface-appropriate a control, the greater the enjoyment and encouraging exploration the control will be. The ideal Surface experience will feel to the user to be completely free of indirect controls.

- Be creative in finding ways to eliminate unnecessary controls. Part of the NUI paradigm is the creative challenge of reducing “interface clutter” and finding innovative ways to let content be the interface itself. For example, look for ways that object recognition can facilitate authentication or text input. Think about ways that users can trade and shuffle controls between themselves to reduce duplicate controls: for example, a shared set of options at the center of the application rather than individual controls for each user. Be bold, reductionist, and clever.

In most instances, the controls in the Surface SDK can be easily adapted to suit new needs. It is important that, when considering the existing controls for various uses, you limit your thinking to the abstract role and purpose of the control, and that you not be inhibited by the particular look and feel or functional mapping included in the Surface SDK and sample applications. To that end, we now examine in detail the controls included in the Surface SDK.

In the following sections, we will review three types of controls: first, those included in the Surface SDK which have been built specifically for development with Surface. Next, we will review the controls which have been migrated from our underlying framework, WPF, and modified to work correctly with Surface. Last, we will describe a control which we have developed for the Surface which may be of interest to you, but which has not yet been made available. It will be useful to you for design inspiration, or as a template for developing your own, similar control.

8.3.4 ScatterView



Definition

A freeform view of content where multiple users can sort, move, resize, rotate, and organize content simultaneously. A ScatterView is comprised of the ScatterView area and multiple ScatterViewItem controls. Each ScatterViewItem can be individually grabbed and simultaneously moved and rotated with one finger. Alternatively, multiple contacts can be used to simultaneously rotate, move, and scale an item.

Proper Usage

Use ScatterView when there are between 2 and 40 pieces of content where interaction should afford multiple simultaneous users. If there are more than 40 pieces of content consider using a [Stack or Scroller](#), or presort content into Stacks so as not to overwhelm the users.

Detailed Description

Use a dark background as the ScatterView for the content. Represent the data in ScatterViewItem controls and arrange the items in a grid, randomly distributed, or in one or multiple Stacks. All ScatterViewItem controls should be able to be rotated, moved, stacked, and resized simultaneously by multiple users.

Initial Views:

Random	Grid	Library
Content cards are presented with random rotation and position	Content cards are presented in a grid with one orientation	ScatterViewItem controls presented in one stack and can be dragged onto the ScatterView for interaction

Choosing among these initial views depends very much on the application and context. Grid view, for example, should be used sparingly: the orientation of the items is not supportive of a 360 UI or multi-user scenario.

ScatterView Properties

The ScatterView is static and should not move if tapped or dragged. Items should not be able to be thrown off the ScatterView. Items should behave in an intuitive way and should bounce off the edge of the screen. The ScatterView should not be zoomed or resized unless your interaction metaphors absolutely necessitate it.

ScatterViewItem Properties

Data is represented within ScatterViewItem controls. Typically these items are rectangular, but can be built to any planar shape.

- **Activation:** touching on the ScatterViewItem should typically activate it. When activated the item should appear to float off the ScatterView. This is shown by a slight increase in size and a larger offset in shadow. The grow and shadow transitions should start slow and end slow. As the object transitions from its non-selected state a slight shimmer should play across it.
- **Depth:** there are two levels to ScatterView: active and stage. As soon as a ScatterViewItem is touched it comes to the top of any stack and moves up to the active level. After the item is released it will drop back down to the stage level after .4 seconds, or after an interval specified by the designer. Items that are recently dropped back down to the ScatterView stay at the top of any content until another ScatterViewItem is dropped on top.
- **Manipulation:** ScatterViewItem controls respond to touches in a natural and intuitive way. Multiple users can rotate, resize and move items simultaneously. ScatterViewItem controls can be flicked and as they bounce back to the ScatterView they should slow down and stop. Items can be spun as they are thrown. Dragging on the edge of the item enables single finger rotation. Dragging in the middle allows the item to be moved without any rotation.

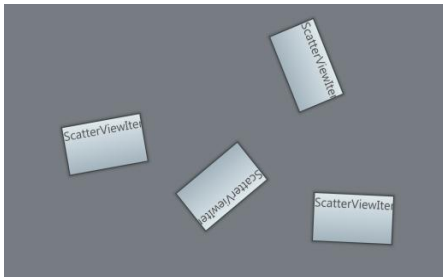
There are three states of ScatterViewItem controls which can be easily assigned varying looks. In the table

below, find a description of each of the states of a ScatterViewItem, including a description of the state, as well as the look and feel provided by default in the Surface SDK.

Enabled

This is the default state for a ScatterViewItem. The user is not in contact with the item, but it is available for input.

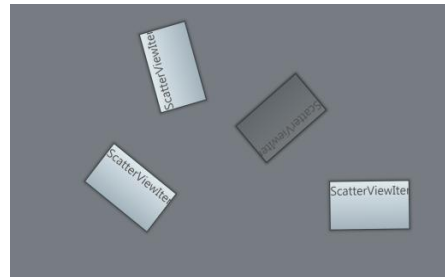
- Content is rendered in a frame with a small shadow on the stage



Disabled

The ScatterViewItem is not available for input. If the user contacts the item, no action is taken.

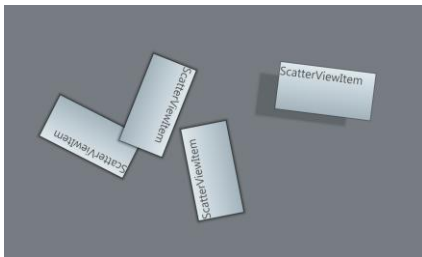
- The content is made darker (grayed out)



Activated

The user contacted the Surface on the ScatterViewItem. The item is brought to the front of the Z-order.

- The item grows in size
- The shadow beneath the item also grows



Multi-Touch Behavior

When interacting with a ScatterViewItem, multiple inputs are accepted. When a single contact is made to the ScatterViewItem, the item will slide around the stage so that the contact point remains at the same position on the item. When a second finger is added, the object will rotate, translate, and scale in order to keep the contacts on the same points within the item. When additional contacts are made, an average is performed so that the approximate same relative position of the contacts and points is maintained.

Benefits & Value

The ScatterView provides a simple mechanism for quickly producing applications which embody the Surface experience. The ScatterView and ScatterViewItem controls provide an empty shell into which you must place content. This content can be as simple as a static image or video, or as complex as a collection of multiple controls, such as a control panel of buttons, checkboxes, and sliders. Items scattered around the stage promote a more social and free-form interaction. Additionally, scattered items create a more approachable and friendly feel as the messy appearance is more human, forgiving, and playful in comparison to the precise and computer feel of grids and tables.



Figure 8.12: A comparison of two example designs to show differences between a gridded, one-person design versus a more freeform social design.

Customization

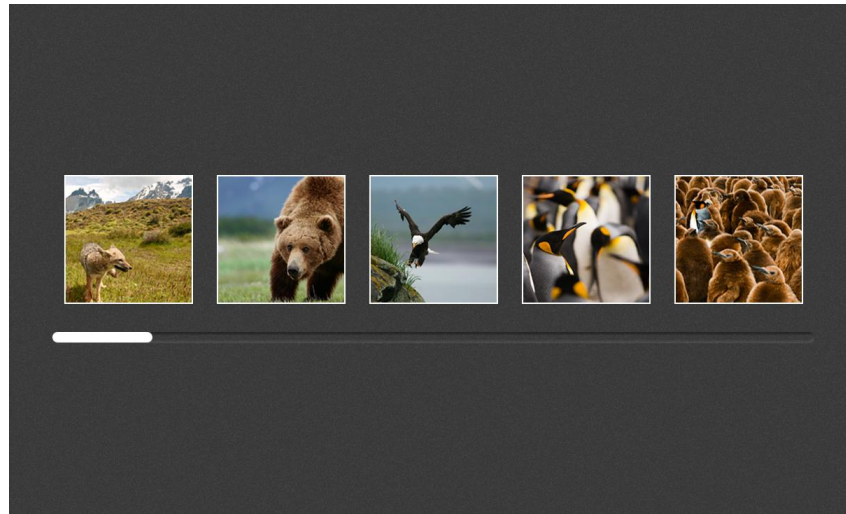
Customization of the ScatterView comes from populating the ScatterViewItem controls with multiple varying types of content. Just about any type of content is possible in the items, from media to collections of Surface SDK controls.

Design Recommendations

- Be smart about default orientation. Ensure that any user, on any side of the screen, can have at some portion of content oriented towards them by default.
- Enable passive learning through animation to show how ScatterViewItem controls can be used. Animating ScatterViewItem controls into place when they instantiate will draw the user's eye to them, and give the user strong clues as to how they may be interacted with. Having seen some items move and rotate onto the stage, dragging and rotating those same items is an easy inference to draw for most users.
- Test for minimum and maximum sizes. Thoroughly test all content and controls held within ScatterView for visibility, legibility, line weight, and brightness at all possible sizes and angles. Inevitably some visual elements will need to be revised based on how they render in ScatterView, or perhaps the minimum and maximum sizes of the objects may need to be revised to maintain visual clarity.
- Content is king. No matter how depth is rendered in ScatterView, it should still be secondary to the content. If testing reveals that depth effects are becoming at least as compelling as the content itself, it is probably wise to pull back on the depth-rendering complexity and let the content shine through.
- When more information needs to be shown in the activated state consider enabling users to flip

the item over and reveal its back. This back side can offer additional information, and during the flip animation the item can change size to better accommodate the type or quantity of information. Additionally, on the back of the item there will need to be a place where the user can flip it back over. For example: when a user taps on a music album, it flips over to reveal a SurfaceListBox with the music tracks from that album. Touching the header of the list flips the item back over. This behavior is not included in the Surface SDK, and must be built by you.

8.3.5 SurfaceScrollView



Definition

This control offers a scrollable area that can contain other visible elements. It is a container to display content that doesn't fit into the desired area. A SurfaceScrollView lets the user directly scroll the content or, if the designer has enabled them, to scroll the content via SurfaceScrollBar controls (one or both of a vertical or horizontal SurfaceScrollBar, as required). Items in a scroll view can be textual, iconic, or can be other types of SDK controls.

Proper Usage

A SurfaceScrollView is intended to provide an easy mechanism for 1D or 2D scrolling of a piece of content which might not otherwise fit in the prescribed area. Use only when you have more content than can be shown at one time on the screen or in the window.

Detailed Description

SurfaceScrollView is a container, which allows the scrolling of underlying content: it can be thought of as a window into a larger content area. The content can vary widely from application to application: it can be as simple as a series of images, or a single continuous area, such as a dynamically updated map.

Interaction with a SurfaceScrollView depends on parameters defined by the designer. It can be constrained to one-dimensional horizontal or vertical scrolling, or enabled for 2-dimensional orthogonal scrolling (mix of horizontal and vertical).

Scrolling in the SurfaceScrollView can include inertia: fast scrolling gestures carry on past the release of the contact, with the view eventually coming to a halt with an apparent friction.

The constraints of the SurfaceScrollView are conveyed to the user through elasticity: scrolling past the

bounds of the viewable area will eventually be stopped by the control. When the user releases the contacts, the view will “bounce” back so that only viewable content is shown.

Multi-Touch Behavior

When interacting with a `SurfaceScrollView`, multiple inputs are accepted. When a single contact is added to the `SurfaceScrollView` the `SurfaceScrollView` starts tracking 1:1 with any movement of the contact. If the movement is constrained to one axis the `SurfaceScrollView` will track 1:1 with the contact within that constraint. When an additional contact is added to the `SurfaceScrollView` the average movement of the two contacts will be transferred to the `SurfaceScrollView`. The same applies to more than 2 contacts; the overall movement is averaged and sent to the `SurfaceScrollView`.

Guidelines

- Use `SurfaceScrollView` when the content will not fit within the desired screen region. If the content will fit without the need for scrolling, do not utilize a `SurfaceScrollView`.
- Carefully order the content within the `SurfaceScrollView`. It is important that the user always understands where they are within the content space. Providing ordering of the content is one way to achieve this.
- Use `SurfaceScrollBar` controls only when the content in the list will likely be longer than three times the visible area of the list. If the list’s content can be shown in 3 or fewer pages, use the direct manipulation technique of panning the list.
- Don’t use `SurfaceScrollBar` controls on the entire window. `SurfaceScrollBar` controls are not recommended when contents runs outside of a view area on multiple axes.

Benefits & Value

The `SurfaceScrollView` provides for an easy, direct-manipulation of a large piece of content.

Look & Feel Customization

The look of the `SurfaceScrollView` is determined almost entirely by the content, which can be virtually any type of image or SDK control. The `SurfaceScrollBar` controls displayed on the `SurfaceScrollView` (if enabled) can also be customized: see the `SurfaceScrollBar` controls section of this document for further details.

Behavior Customization

The designer can elect to allow direct manipulation to scroll content, can require the use of `SurfaceScrollBar` controls, or can use a combination of the two. The designer can also customize the elasticity of the `SurfaceScrollView`, which controls the behavior as the user reaches the edges of the content.

Two types of scrolling are supported natively: horizontal or vertical constrained (1D scrolling) and a combination of horizontal / vertical scrolling (2D scrolling).

Design Recommendations

Some of the important recommendations with respect to `SurfaceScrollView` relate to the sliders which might be used to scroll content:

- Design compact, subtle `SurfaceScrollBar` controls that are easily touchable. Ensure that hit states for thumbs are no less than 38 pixels wide. They should be at least this size when pressed, but

may appear visually as small as 7 pixels wide in their enabled (but not pressed) state. Images of the default appearance for each of these states can be found below in the SurfaceScrollBar controls section.

- Design for the grow-upon-touch paradigm. Ensure that sliders and thumbs have larger states that are visible below a user's finger. Slider thumbs could simply scale in size, or have auras around them, but their active states must be visually very clear. Be sure the enlarged state is harmonious with the rest of the slider design and the broader design vocabulary of the experience.
- The contents should take prominence and lead the interaction. A SurfaceListBox should be the user's primary control, and the SurfaceScrollBar should be relegated to showing the user's current position within the total list of options.
- Enable passive learning through animation to show how SurfaceScrollViewer controls can be used. Animating a SurfaceScrollViewer's contents into place when it instantiates will draw the user's eye to it, and give them strong clues as to how it may be interacted with. Having seen some content scroll out of view, flicking the SurfaceListBox to view that content is an easy inference to draw for most users.

Example

A SurfaceScrollViewer is used in the Launcher application to allow users to easily scroll among the available applications. In this case, scrolling has been constrained to the horizontal, and SurfaceScrollBar controls have been disabled. The content within the SurfaceScrollViewer in this case is a series of SurfaceButton controls.



Figure 8.13: Launcher. The content of the SurfaceButton controls are animations, and their size is a function of their on-screen location.

8.3.6 TagVisualizer



Definition

The TagVisualizer control reacts to one or more Byte Tagged physical objects that are placed on the Surface screen by creating and displaying TagVisualization objects. A TagVisualization object represents the actual user interface that users see. A TagVisualizer control automatically tracks the motion of a Byte Tagged object and moves the TagVisualization object with the physical object.

Proper Usage

Use when you wish to have your application present a visualization in response to Byte Tagged objects. Such visualizations are useful in presenting a virtual interface to the physical object. This enables information and/or controls to be presented in tight association with the tagged object. Additionally, such visualizations provide users with explicit touch targets and feedback when interacting with the object's virtual interface (e.g., dragging or moving content into the physical object).

Detailed Description

The TagVisualizer does not provide a default visualization—it is the responsibility of the designer and developer to build this visualization. In essence, it provides a container into which content can be placed that should be displayed surrounding a tagged object placed on Surface. That content can be any type of object, including text, graphics, and other UI controls.

Benefits & Value

Provides an easy development tool for quickly building applications which utilize tags.

Look & Feel Customization

The particular visualization is completely customizable.

Behavior Customization

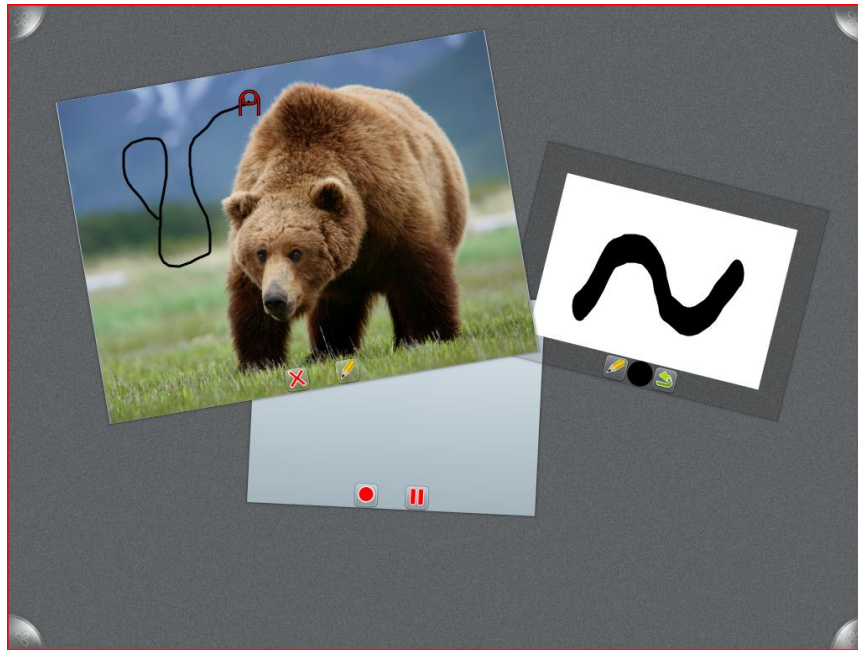
The TagVisualizer provides the behavior of a control which tracks tags placed on Surface. The particular

behavior of the content of TagVisualizer for any given Byte Tag is completely customizable. In particular, the behavior of the system when a Byte Tag is removed, how long a Byte Tag must be absent from the Surface screen before it is considered to have been removed, and whether or not the Byte Tag orientation is a parameter to the visualization is all customizable.

Design Recommendations

- In general, when designing TagVisualization controls, keep in mind that the tagged object will occlude the region immediately around the Byte Tag. Keeping this in mind, we recommend that you keep track of the physical parameters of each tagged object, so that your visualizations can respond appropriately to each object individually.
- Not all tagged objects need or should have associated visualizations. Obvious examples may include physical game pieces (i.e., chess pieces).
- When a tagged object is rotated, consider what should happen to its associated visualizations or any other elements of the overall UI. Depending on the object itself, its purpose, and context consider rotation of other elements to align orientation between physical object and virtual content. For example, rotating a tagged cell phone may cause only the contents of its TagVisualizer to rotate, or the entire screen UI could also rotate.
- If it is valid for users to move other digital objects or content into a TagVisualizer (e.g., dragging a digital photo into a physical camera), then consider the following:
 - Provide visual feedback of valid destinations: as users move a valid object / content from elsewhere on the screen, highlight or bring the visualization to life to indicate that it is a valid destination or target;
 - Provide visual feedback of the drop: once the object has been moved sufficiently into contact with the visualization animate the object's transition "into" the physical object as necessary feedback; do not simply have the object disappear as it is important for users to suspend disbelief and see the digital object "go into" the physical object.
- Enable users to flick or throw digital content toward the TagVisualizer.
- Provide a discoverable affordance to help the user understand that a tagged item can be placed on the Surface screen, and where it should be placed.

8.3.7 SurfaceInkCanvas



Definition

A `SurfaceInkCanvas` control receives input as the user moves a contact or contacts over the control and displays ink in response. During contact movement, the control creates strokes, which can be manipulated either programmatically or based on user input. Once a `Stroke` object is created, the `SurfaceInkCanvas` enables users to delete it. The Photo Paint application, pictured above, demonstrates the `SurfaceInkCanvas` control.

Proper Usage

Use this control to allow users to draw on the Surface using their hands and other objects.

Multi-Touch Behavior

When interacting with a `SurfaceInkCanvas`, multiple inputs are accepted. Each new contact starts a new stroke. A stroke finishes when the original contact lifts up from the canvas.

Benefits & Value

`SurfaceInkCanvas` provides an easy mechanism to allow the drawing of strokes via direct user input. `SurfaceInkCanvas` can also contain other elements, which makes it ideal for allowing users to mark up visual items, such as editing pictures or highlighting text in a newspaper article.

Customization

The `SurfaceInkCanvas` is, in essence, a blank canvas. Content can be added programmatically, and basic behaviors changed. The response to different touch types can be customized, so that it responds differently to touches by fingers, larger objects, and tagged objects.

Of particular interest is whether or not the canvas is enabled to match the shape of the contact in producing strokes. In one mode, the precise shape of the contact area to produce the stroke. In the other mode, the `SurfaceInkCanvas` uses a fixed brush shape which roughly matches the size of the contact area.

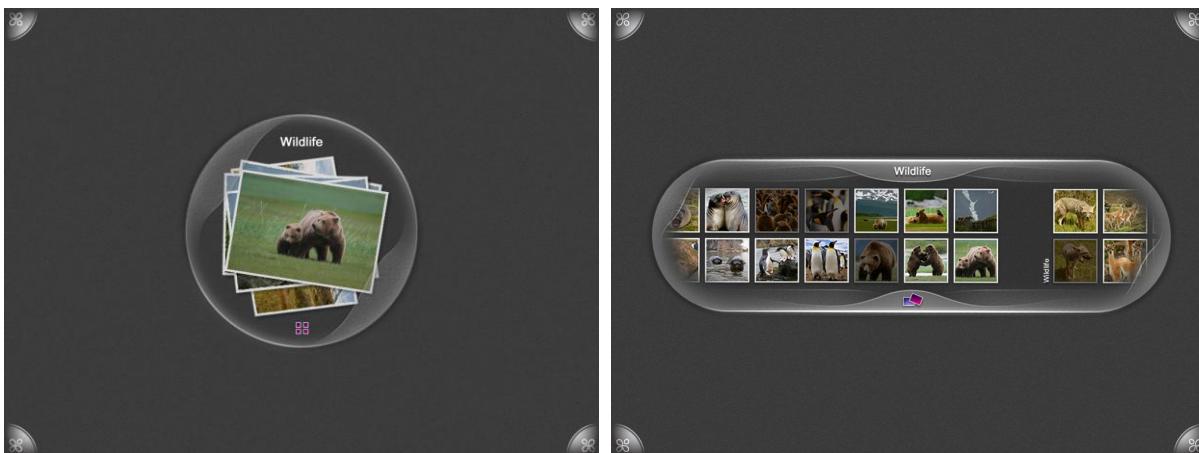
In choosing between these modes, consider whether your users will require the precise shapes of the contacts, or whether a more consistent, “brush” appearance to their strokes is more desirable.

Design Recommendations

If you wish to allow users to manipulate the position, size, and orientation of a SurfaceInkCanvas, it can be placed within a ScatterViewItem. A border region around the outside of the Canvas needs to be included to serve as a handle for touches to exert this control. To ensure there is sufficient space to allow for full control over these parameters, this border region should be at least 43 pixels in size.

8.3.8 Stack & Scroller

This control is not currently in our Surface SDK, but has been designed and is in use with existing Surface applications. It is being shared as a recommended control for your consideration. This control must be designed and developed by you.



Definition

Stack and Scroller each represent a container holding a collection of objects. Those objects have both an order and a category. The Stack and Scroller differ in their visualizations: a Stack is a virtual stack of objects, one placed atop another, which users can sort-through and reorder. Scroller, in contrast, is laid out in a horizontal arrangement of content, which users can scroll horizontally. These can be implemented as separate controls, or different visualizations of the same control, with the Stack thought of as a collapsed view of the Scroller. Both controls are fully manipulatable; users can move, rotate, resize, and flick them about, and users can move individual pieces of content out of or into them.

Proper Usage

The Stack and Scroller are each suitable for organizing groups of content. This is particularly useful when there is too much content making a scattered view overly cluttered and unusable. If available, meta-information, such as content types, dates, or other categorical measures can be used to easily organize the content.

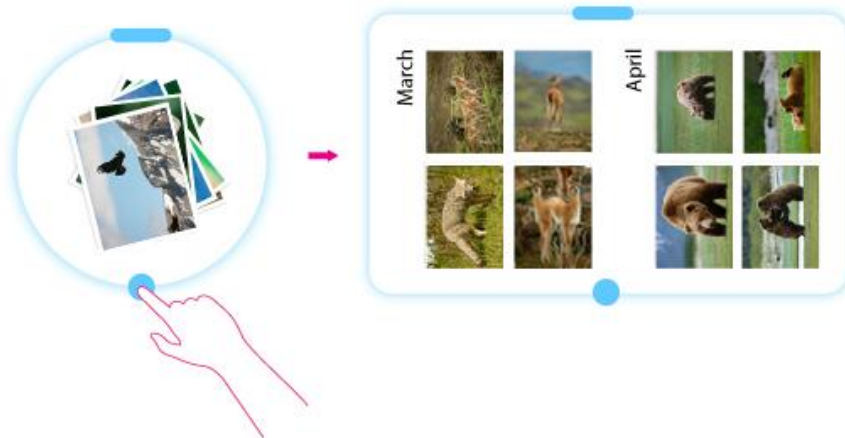
For example, if a lot of user content is being downloaded or imported from somewhere, it may be best to have it all appear first within a Stack or Scroller. The added benefit is that users can begin to orient to and play with the first set of downloaded content while the rest continues to load.

Detailed Description

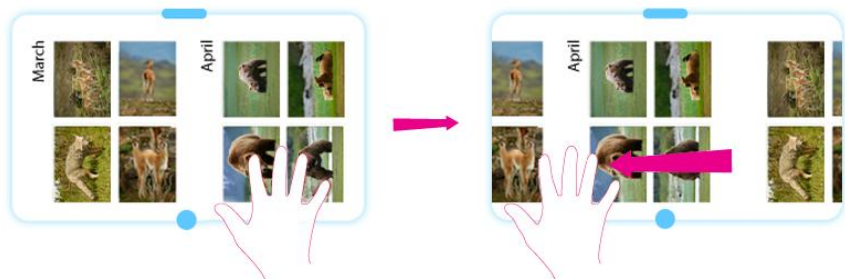
The vertical ordering of content in a Stack can be easily changed. To move the top item to the bottom of the Stack, the user flicks it. To select an item to be moved to the top of the Stack, the user can simply touch the protruding portion.



The groups can be viewed as a Stack or a Scroller. The default view is stacked, but the user can change the view to a Scroller by touching the view changer located on the pod's aura, or by stretching the pod horizontally to reach Scroller view. The benefits of the Scroller view are that more of the content is visible at one time, and it is organized by category and labeled accordingly. In the figure below, the categorization of stack content is used to create a list sorted in chronological order.



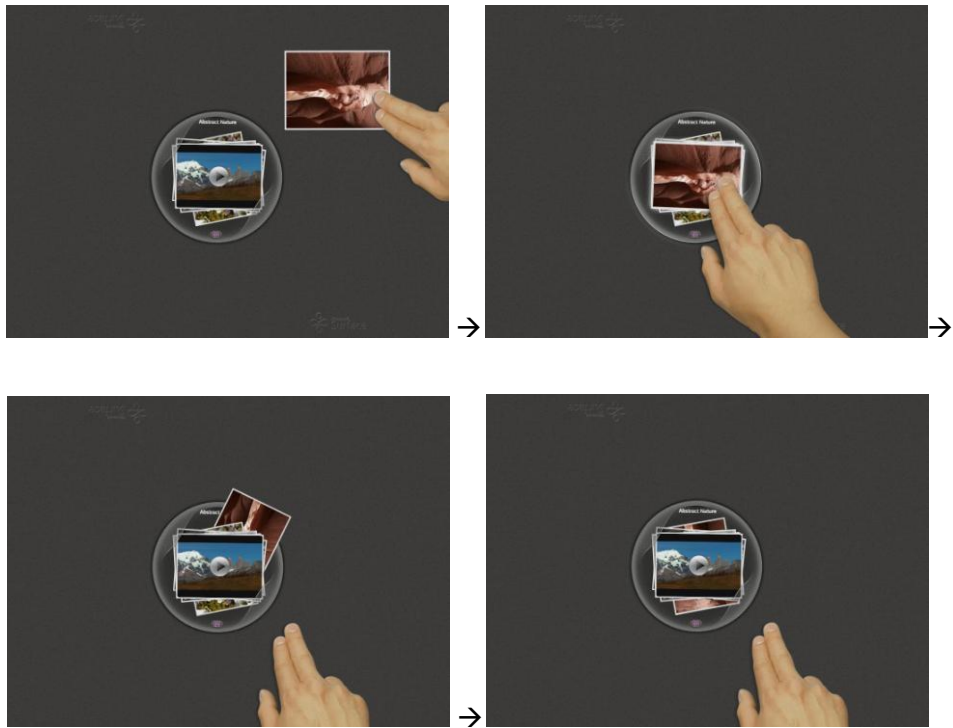
The user touches the button at the bottom of the control to change the view from Stack to Scroller. Once in a Scroller, the content can be scrolled left or right. This is implemented using the SurfaceScrollerView, described previously.



Additionally, content can be moved out of a Stack and Scroller. For a Stack, users can simply drag the top item out onto the application's background or stage. For a Scroller, users must drag an item up or down to move it out (left or right movement will cause scrolling).



Adding an item or piece of content to either control is simply a matter of dragging or flicking it onto the control. To ensure that it is truly the user's intent to add the item to the control, it is best to have a majority of the content overlapping the Stack or Scroller in order to add it. If the Stack or Scroller's content is organized or sorted, then an item added to it may animate "into" it and go out of sight as it positions itself appropriately to accommodate the organization.



When working with either the Stack or Scroller, the user can touch the drop-down menu located at the top of the Scroller to jump to a category. When performing this selection in the Stack, any content of that category which has been removed from the Stack and placed elsewhere on the stage is brought to the top of the Z-order of the ScatterView. Additionally, as users scroll through the content of these containers, the drop-down menu will automatically change its value to reflect the current category of content being displayed.



Multi-Touch Behavior

Stacks and Scrollers are made up of several other controls: items of content, SurfaceScrollView controls, buttons, and custom controls. The multi-touch behavior of these controls is as described elsewhere.

Generally, the multi-touch behavior of Stack and Scroller follows the principle of modeless interaction: when dragging a Stack, for example, another contact can be used to grab or re-order the photos.

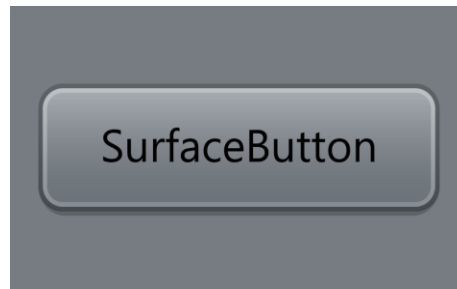
Customization

The concepts demonstrated in these custom controls can be applied in a wide variety of applications. Grouping is a common function in many tasks, and providing these grouping mechanisms will enable users of your applications to work with content in a fast and intuitive way.

Design Recommendations

- Stacks and Scrollers should always be at the top of the application's background or Z-order.
- When an item is removed from the Scroller, consider whether a ghosted version should be left behind. Users seeing such a ghosted version realize that this is where the item resides normally, but it is out on the stage somewhere. Touching this ghosted version should cause the real version to float up to the top of the stage's Z-order (facilitates finding).
- If users manually stack or pile content then the application should automatically create a Stack around that pile. We recommend doing this when the user adds the 3rd item onto his/her pile. In such cases, a user Stack or Scroller has been created. Such user-created containers can be temporary in that once the user removes the last piece of content the container can disappear. Also, when a content item is removed from a user-created container, it should not leave behind a ghosted version. Contrast this to a system-created container, which represents a more permanent collection (e.g., photos downloaded from a website).
- As containers, Stacks and Scrollers are a great way to manage and organize potential screen clutter. If more than one of these objects will be on the stage at any given time, they too may eventually contribute to screen clutter. Also, users may want to enjoy playing with content and avoid having items accidentally added to these containers. Therefore, enable users to temporarily move or flick these containers partially off-screen. Enforce a minimum amount to remain on stage and be visible; certainly enough for users to effectively touch it and bring it back on screen. Also, avoid having items becoming inaccessible if moved partially off-screen under an Access Point.

8.3.9 SurfaceButton



Definition

A SurfaceButton is a region of the screen which has been labeled with a command. When touched, the command labeled on the button will be run.

Proper Usage

SurfaceButton controls are appropriate when used to initiate an immediate action. In keeping with the principle of Direct Manipulation, it is appropriate to utilize SurfaceButton controls only when a direct and physical interaction is not apparent or appropriate to initiate an action.

A SurfaceButton's associated action or function is started when the user's finger lands on a button and is removed from the Surface while within the confines of that button. It is important that this action not be initiated before the user's finger is removed, and not occur at all if the user's finger slides off of the button before being lifted from the Surface. This gives the user the opportunity to reconsider and possibly to cancel their action before continuing. An example of the proper usage of a button on the Surface is shown in the next figure: the SurfaceButton occupies the entirety of the top of the album, blending it among the content. When the user touches the top of the album, it grows, as if moving towards the user, acknowledging their contact. When the user removes their hand, the album flips to reveal the cover art.

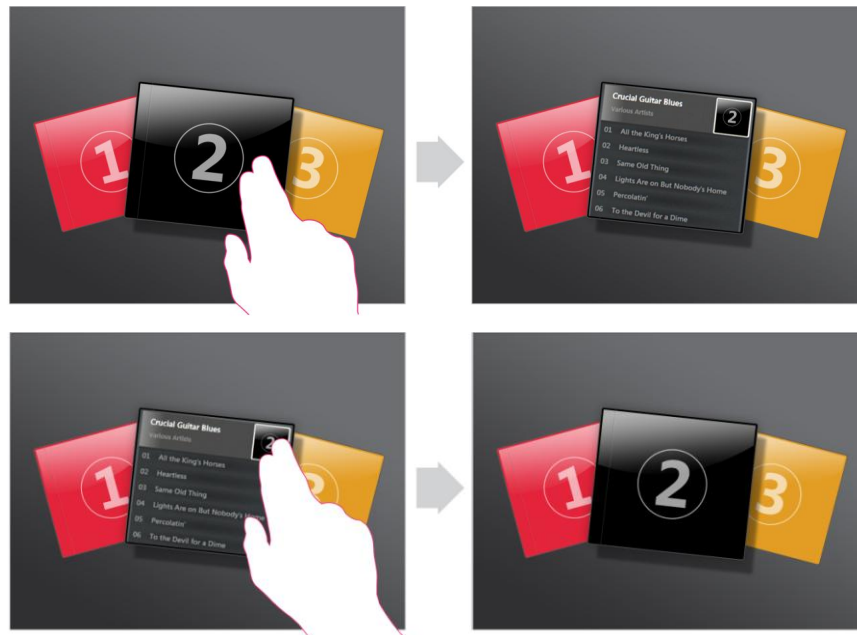


Figure 8.14: In our Music application, tapping the cover art flips the album to reveal the song list. Tapping the SurfaceButton at the top of an album song list causes the album to flip back over, showing the cover art again.

Detailed Description

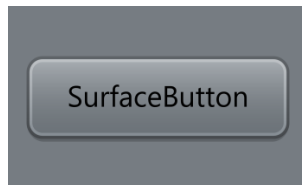
SurfaceButton controls are intended to complete immediate actions, such as beginning a process or submitting a form, and not for other purposes, such as setting parameter values or states.

The labels on a SurfaceButton can be textual or iconic. There are four states of the SurfaceButton which can be easily assigned varying looks. In the table below, find a description of each of the states of a SurfaceButton, including a description of the state, as well as the look and feel provided by default in the Surface SDK.

Enabled

This is the default state for a button. The user is not in contact with the button, but it is available for input.

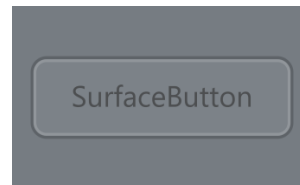
- No glow effect on the border
- Text is the default color
- Background gradient is the default color



Disabled

The button is not available for input. If the user contacts the button, no action is taken.

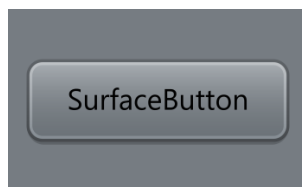
- No glow effect on the border
- Text is darker (grayed out)
- Background gradient is removed



Captured

The user contacted the Surface on the button, and then dragged away from the button, or the user contacted the Surface outside of the button, and then dragged over it. Generally this state should not be used to initiate actions.

- No visual effect.



Pressed

The user initiated contact on the button. Generally this state should not be used to initiate actions.

- Glow effect on the border
- Text color is grayed
- Background color is grayed



Multi-Touch Behavior

When interacting with a button, multiple inputs are accepted. The control state changes from the default

state to the pressed state when the first contact touches the button. When subsequent contacts are added to the button no additional action is initiated. The action associated with the SurfaceButton is initiated when the last contact captured to the control is removed from the Surface while still inside the control.

Benefits and Value

The SurfaceButton provides a physical manifestation of an abstract or metaphorical action, one for which a physical action is not otherwise apparent or appropriate. It provides a focal point for interaction, with a clear and well understood metaphor.

Customization

Designers are able to customize two aspects of this control: its interactive behavior and its visual look & feel.

Behavioral Customization

The SurfaceButton's invocation event can be changed. The moment of invocation of the command during a user's interaction with a SurfaceButton can be customized. Normally, the command is run when the user releases the button, after having first pressed it. This is the normal and preferred behavior. Alternatively, the command can be invoked when the user first presses down on the button, when the user's finger slides off the button, or when the user's finger slides over (hovers) over the button without first pressing it down. In most circumstances, activation on release is preferred, because it gives the user the opportunity to cancel their action by sliding off the button. In some applications, such as the Grand Piano sample application shown in the next figure, a modified behavior is necessary to achieve the desired result.



Figure 8.15: The Grand Piano sample application. The keys are SurfaceButton controls with their behavior modified to also activate when the finger slides onto them, creating a more realistic keyboard.

Visual Customization

All visual elements of the button can be changed: its shape, its colors, its visual appearance in each of the visual states, and its transition animation can all be customized. The above image shows the keys from our Grand Piano sample application. Each of these keys is a SurfaceButton with its shape and visual appearance modified.

It is important that any customization of appearance continues to afford touching by the user. While the piano keys in the above figure do not resemble default SurfaceButton controls, it is still clear to the user that they are interactive.

Another example of custom buttons is Launcher, shown in the next figure. In this case, the content of the buttons has been modified to show an animation of the application they represent.



Figure 8.16: Launcher. The content of the SurfaceButton controls is an animation, and their size is a function of their on-screen location.

In addition to the content animation, the buttons have been placed inside of a SurfaceScrollView control, and the size of the button and space between buttons has been tied to its position on the screen. This creates a focal point, as well as allowing for larger buttons without the need for the user to first manually select the button from the list.

Design Recommendations

Behavioral Design

- The interactive behavior of a SurfaceButton is prescribed by the Surface SDK. Although it would be possible to build a replacement control which mimics some subset of the behavior provided, it is highly recommended that this not be done. In nearly all cases, desired behaviors and appearances can be achieved by replacing the look and feel of the button, rather than replacing it outright.
- Display appropriate graphics if the result of executing a SurfaceButton isn't instantaneous. Without feedback, users might assume that the selection didn't happen and press again. While this is true of traditional computer interfaces, it is especially true on the Surface, in which the vision system produces a higher error rate for input recognition. Ideally, these graphics would occur within the confines of the SurfaceButton itself, in keeping with our principle of direct manipulation.

Visual Design

- Ensure clear legibility of button labels. It is easy to overlook the fact that users may need to press buttons that may not be perfectly oriented towards their position around the Surface screen. Likewise, button controls are usually embedded in (or represented as) other objects or content that can be moved or rotated away from their original orientations. Make sure that you follow proper typographic guidelines when typesetting button labels (see [Typography](#)): use strong sans-serif typefaces, make button labels at least 12 pixels in height, and that the label is legible from all angles and sizes (if the button or its parent object is scalable).
- Use richly formatted content as buttons themselves. Buttons need not always be discrete visual objects, so always look for opportunities to use content itself as a button. Be sure to balance this simplicity of form with clarity of function; visual minimalism of a button must always be balanced with how predictable or natural its response or result will be.
- Buttons must be the appropriate minimum size for easy interaction. Be sure that all button

controls are no less than 30 pixels wide. This is the minimum size of a touch gesture. The larger the button, the easier it will be to touch, and the easier its label will be to read.

- Volume is good, but avoid directional button movements. While it is appropriate to have buttons that are volumetric, to encourage touch and interactions, do not make buttons appear to physically depress below the plane of the Surface screen. Also avoid any visual movements with any sort of perspective; some users will see this from the wrong angle and it will look incorrect, disrupting suspension of disbelief. Illuminating from within is a more successful toggle-state or reaction indicator.
- It is generally preferred that controls be embedded with the content of an application, rather than appearing alongside it as a separate visual element. The following example, taken from our own internal designs, illustrates a preferred implementation of a SurfaceButton.

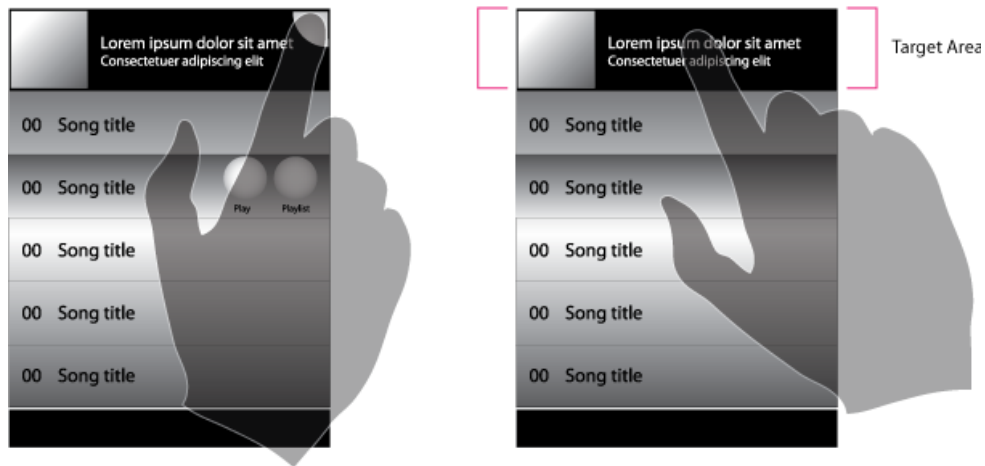


Figure 7.17: Left (non-preferred): a small button, located in the top-right corner, is used to ‘flip’ an album cover. Right (preferred): a larger area is used as a SurfaceButton, without the additional graphical area for the button.

Alternative Design Patterns

The wide array of alternative designs made possible by providing alternative visuals for Surface SDK controls means that just about any look and feel can be achieved for SurfaceButton controls. In general, the use of SurfaceButton controls for many situations is generally non-preferred for a Surface experience. For example, consider the selection of a song to play from an album. Under a traditional scheme, the user might select a song from the list, and then press a “Play” button. A Surface experience would require the user only to select their desired song from the list.

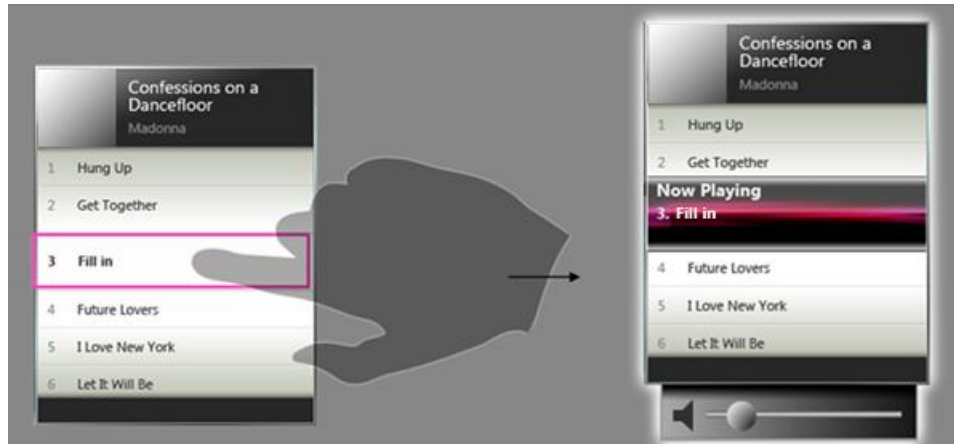


Figure 8.18: Rather than selecting a song and then pressing 'play', the song plays immediately when touched. This removes a level of abstraction.

In general, activation and the mapping of functionality directly onto the content is desired, in favor of separating the object from the SurfaceButton causing it to perform an action.

Examples and Samples

As we have seen, SurfaceButton controls can be modified in their look and feel. The shell invocation buttons, Access Points, are an example of a non-square button with an animation performed when pressed.



Figure 8.19: These buttons are rounded, and show an animation when pressed.

Launcher uses SurfaceButton controls to both demonstrate applications to the user and to allow the user to select an application. These buttons have been modified to include animations demonstrating the functionality which lies within:



Figure 8.20: Launcher from the Surface Shell. Each of the applications is listed with an animation demonstrating its use, residing within the SurfaceButton used to activate it.

The Music application uses SurfaceButton controls to allow the user to control the current track:

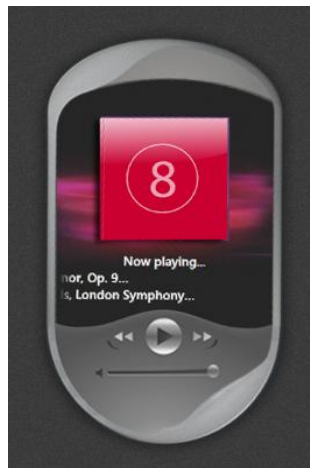
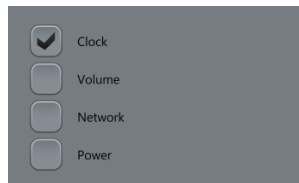


Figure 8.21: The Music application's pause and track change controls are instances of SurfaceButton controls.

8.3.10 SurfaceCheckBox



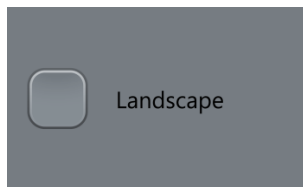
Definition

A SurfaceCheckBox allows users make a decision between two clearly opposite choices. The SurfaceCheckBox label indicates the selected state, whereas the meaning of the cleared state must be the unambiguous opposite of the selected state. Consequently, SurfaceCheckBox controls should be used

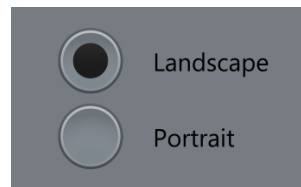
only to toggle an option on or off or to select or deselect an item.

Proper Usage

- Is the SurfaceCheckBox used to toggle an option on or off or to select or deselect an item? If not, use another control.
- Are the selected and cleared states clear and unambiguous opposites? If not, use [SurfaceRadioButton controls](#) so that you can label the states independently.
- When used in a group, does the group comprise independent choices from which users may choose zero or more? If not, consider controls for dependent choices, such as SurfaceRadioButton controls and SurfaceListBox.
- When used in a group, does the group comprise dependent choices from which users must choose one or more? If so, use a group of SurfaceCheckBox controls and handle the error when none of the options are selected.
- Would a SurfaceRadioButton be a better choice? Where check boxes are suitable only for turning an option on or off, radio buttons can be used for completely different options. If both solutions are possible:
 - Use SurfaceRadioButton controls if the meaning of the cleared SurfaceCheckBox isn't completely obvious.

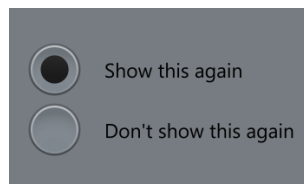


Incorrect: In this example, the opposite choice from Landscape isn't clear, so the SurfaceCheckBox isn't a good choice.

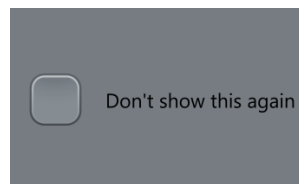


Correct: In this example, the choices are not opposites so SurfaceRadioButton controls are the better choice.

- Use SurfaceRadioButton controls on wizard pages to make the alternatives clear, even if a SurfaceCheckBox is otherwise acceptable.
- Use SurfaceRadioButton controls if you have enough screen space and the options are important enough to be a good use of that screen space. Otherwise, use a SurfaceCheckBox.



Incorrect: In this example, the options aren't important enough to use SurfaceRadioButton controls.



Correct: In this example, a SurfaceCheckBox is an efficient use of screen space for this peripheral option.

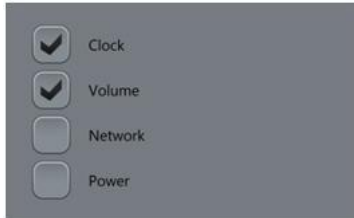
- Use a SurfaceCheckBox if there are other SurfaceCheckBox controls on the window.

Detailed Description

SurfaceCheckBox controls have two primary usage patterns: individual choices and independent choices.



An individual choice: A single check box is used to select an individual choice.



Independent choices (zero or more): A group of check boxes is used to select from a set of zero or more choices. Unlike single-selection controls such as [SurfaceRadioButton controls](#), users can select any combination of options in a group of check boxes.

The labels on a checkbox can be textual or iconic. There are several states of the checkbox which can be easily assigned varying looks. In the table below, find a description of each of these states, including a description of the state, as well as the look and feel provided by default in the Surface SDK.

Enabled: Checked

The user is not in contact with the checkbox, but it is available for input. It is currently set to "True".

- There is no glow effect around the check area
- The box is checked



Captured

The user contacted the Surface outside the checkbox, and then dragged over the button. Generally this state should not be used to initiate

Enabled: Unchecked

This is the default state for the control. The user is not in contact with the checkbox, but it is available for input. It is currently set to "False".

- There is no glow effect around the check area
- The box is not checked



Pressed: Checked

The user initiated contact on the checkbox while it was *checked*. Generally this state should not be

actions.

- No visual effect



Pressed: Unchecked

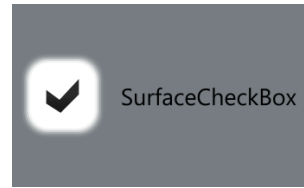
The user initiated contact on the checkbox while it was *unchecked*. Generally this state should not be used to initiate actions.

- Glow effect within the box



used to initiate actions.

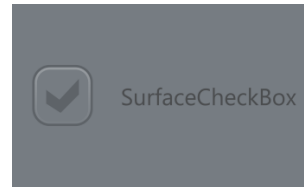
- Glow effect within the box



Disabled Checked / Unchecked

The checkbox is not available for input. If the user contacts the checkbox, no action is taken.

- No glow effect on the border
- Text is lighter (grayed out)
- The checkmark (if present) is lighter (grayed out)



Multi-Touch Behavior

When interacting with a SurfaceCheckBox, multiple inputs are accepted. The control state changes from the default state to the pressed state when the first contact touches it. When subsequent contacts are added to the control no additional action is initiated. Release is initiated when the last captured input is been removed from the control while still inside the control. Release causes the control to toggle between checked and unchecked.

Benefits & Value

The checkbox provides a physical manifestation of an abstract or metaphorical property, one for which a visual state is not otherwise apparent or appropriate. It provides a focal point for interaction with a clear and well understood metaphor.

Customization

Designers are able to customize the look & feel of this control.

All visual elements of the checkbox can be changed: its shape, its colors, its visual appearance in each of the visual states, and its transition animation can all be customized. It is important that any customization of appearance continues to afford touching by the user. While you may create other designs which are not visually similar to the SDK checkbox, it is important that your designs continue to afford touch.

Design Recommendations

- Ensure text label legibility. Be sure that you follow appropriate typography guidelines when

typesetting SurfaceCheckBox labels, especially if the object containing the SurfaceCheckBox control(s) can be freely scaled and/or rotated. Use strong sans-serif typefaces, make button labels at least 12 pixels in height, and that the label is legible from all angles and sizes.

- Reserve illumination, effects, animations, or other ornamentation for touch feedback or selected states. Checkboxes should be minimal, understated, and subtle until touched. When touched, a strong visual reaction is appropriate. A somewhat accentuated state is also appropriate for the selected state of a SurfaceCheckBox.
- Volume is good, but avoid directional button movements. Use volumetric shapes and shading to encourage touch and interactions, but do not make SurfaceCheckBox controls appear to physically depress below the plane of the Surface screen or move in three dimensions. Illuminating from within is a more successful toggle-state or reaction indicator.

Alternative Design Patterns

The wide array of alternative designs made possible by providing alternative visuals for Surface SDK controls means that just about any look and feel can be achieved for checkboxes. In general, the use of checkboxes for many situations is generally non-preferred for a Surface experience.

For example, consider the task of customizing the options on a vehicle. Several options can be selected or deselected to allow the user to customize their car. In a traditional WIMP environment, these might all be selected using checkboxes. The preferred Surface method, in keeping with our design principles, is a more direct-touch and physical interaction.

A design which better conforms to this is sketched in the figure below. The user is presented with physical representations of the various options for the vehicle. They then select to include those options on their customized car by dragging them onto the custom vehicle, which then immediately reflects the addition or deletion of the option from the vehicle.

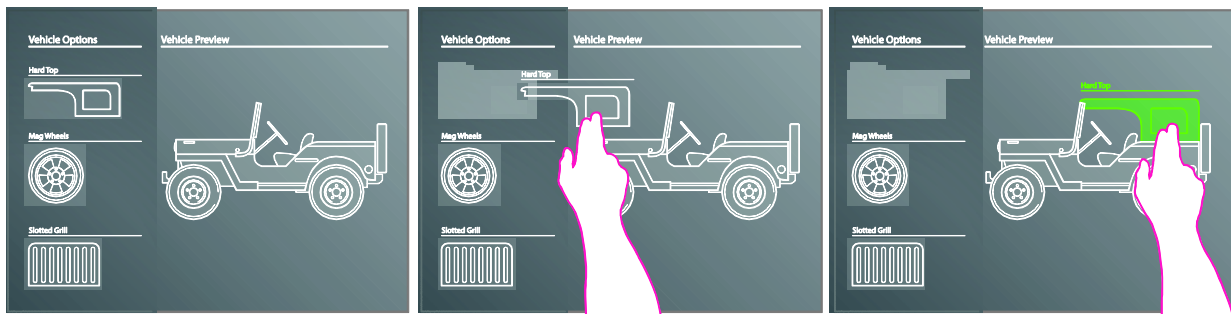
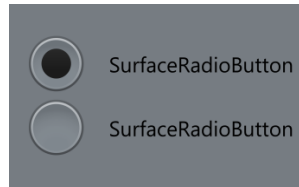


Figure 8.22: An alternative design for selecting features while customizing a car. Features are shown as physical items, which are selected by dragging-them onto the in-production vehicle.

This design can be abstracted to a general pattern of presenting possible choices as physical icons, with selections made by dragging the appropriate item to a selection area. In general, this approach is preferred to SurfaceCheckBox controls because it provides a more natural, enjoyable experience, as described in our design principles.

8.3.11 SurfaceRadioButton



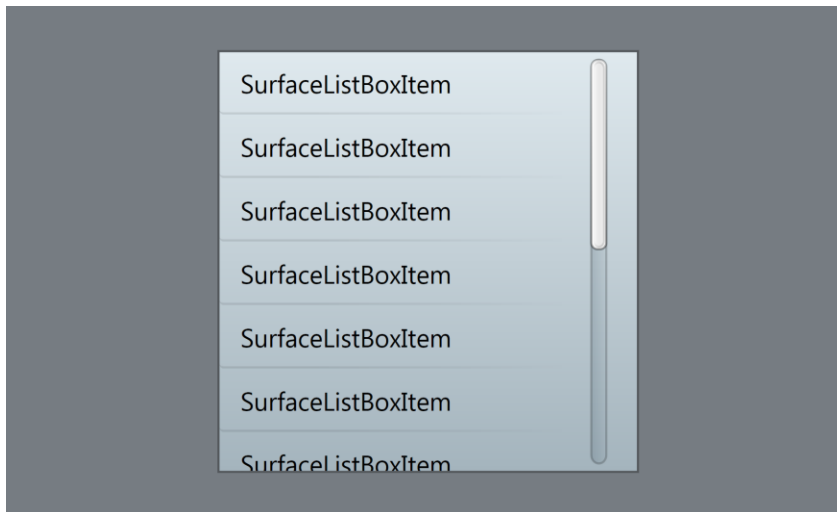
Definition

With a SurfaceRadioButton, users make a choice among a set of mutually exclusive, related options. Users can choose one and only one option. SurfaceRadioButton controls are so called because they function like the channel presets on radios.

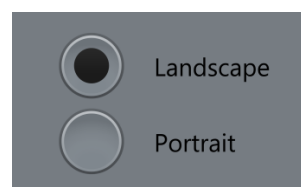
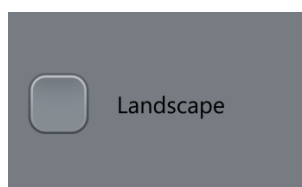
Proper Usage

To decide if this is the right control to use, consider these questions:

- Is the control used to choose one option from a set of mutually exclusive choices? If not, use another control. To choose multiple options, use SurfaceCheckBox controls instead.
- Is the number of options between two and seven? Since the screen space used is proportional to the number of options, keep the number of options in a group between two and seven. For eight or more options, use single-select SurfaceSurfaceListBox



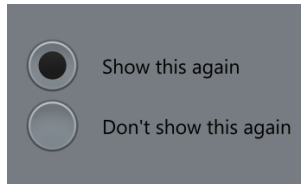
-
- Would a SurfaceCheckBox be a better choice? If there are only two options, use a single SurfaceCheckBox instead. However, SurfaceCheckBox controls are suitable only for turning a single option on or off, whereas SurfaceRadioButton controls can be used for completely different alternatives. If both solutions are possible:
 - Use SurfaceRadioButton controls if the meaning of the cleared SurfaceCheckBox isn't completely obvious.



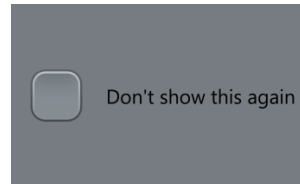
Incorrect

Correct

- In the correct example, the choices are not opposites so SurfaceRadioButton controls are the better choice.
- Use SurfaceRadioButton controls on wizard pages to make the alternatives clear, even if a SurfaceCheckBox is otherwise acceptable.
- Use SurfaceRadioButton controls if you have enough screen space and the options are important enough to be a good use of that screen space. Otherwise, use a SurfaceCheckBox.



Incorrect. In this example, the options aren't important enough to use SurfaceRadioButton controls.

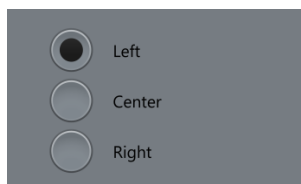


Correct. In this example, a SurfaceCheckBox is an efficient use of screen space for this peripheral option.

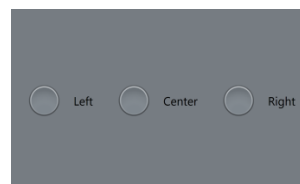
- Use a SurfaceCheckBox if there are other SurfaceCheckBox controls on the page.
- Do the options present program options, rather than data? The options' values shouldn't be based on context or other data. For data, use a SurfaceListBox.
- Are the values non-numeric? For numeric data with a large number of options, use a text box or SurfaceListBox. If the numeric values are contiguous, use a slider.

Guidelines

- List the options in a logical order, such as most likely to be selected to least, simplest operation to most complex, or least risk to most. Alphabetical ordering is not recommended because it is language dependent and therefore not localizable.
- If none of the options is a valid choice, add another option to reflect this choice, such as *None* or *Does not apply*.
- Align SurfaceRadioButton controls vertically instead of horizontally. Horizontal alignment is harder to read and localize.



Incorrect. In this example, the SurfaceRadioButton controls are aligned vertically.



Correct. In this example, the horizontal alignment is harder to read.

- Don't use SurfaceRadioButton labels as group box labels.
- Don't use the selection of a SurfaceRadioButton to:

- Perform commands.
- Display other windows, such as a dialog box to gather more input.
- Dynamically show or hide other controls related to the selected control (screen readers cannot detect such events). However, you can change text dynamically based on the selection.

Detailed Description

SurfaceRadioButton controls are intended to make an exclusive selection from a list. Selecting a SurfaceRadioButton within a group will automatically un-select another SurfaceRadioButton in the same group. There are several states of the SurfaceRadioButton which can be easily assigned varying looks. In the table below, find a description of each of these states, including a description of the state, as well as the look and feel provided by default in the Surface SDK.

Enabled: Selected

The user is not in contact with the SurfaceRadioButton, but it is available for input. It is currently set to "True".

- There is no glow effect around the button area
- The button is selected



Enabled: Unselected

This is the default state for the control. The user is not in contact with the SurfaceRadioButton, but it is available for input. It is currently set to "False".

- There is no glow effect around the button area
- The button is not selected



Captured Selected / Unselected

The user contacted the Surface outside the SurfaceRadioButton, then dragged over the button, or on the button, then dragged off. Generally this state should not be used to initiate actions.

- No visual effect



Pressed: Selected

The user initiated contact on the SurfaceRadioButton while it was *selected*. Generally this state should not be used to initiate actions.

- Glow effect within the box



Pressed: Unselected

Disabled Selected / Unselected

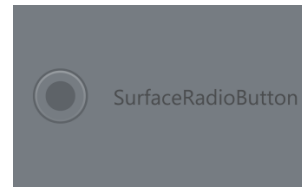
The user initiated contact on the SurfaceRadioButton while it was *unchecked*. Generally this state should not be used to initiate actions.

- Glow effect within the box



The SurfaceRadioButton is not available for input. If the user contacts the checkbox, no action is taken.

- No glow effect on the border
- Text and button area are lighter (grayed out)



Multi-Touch Behavior

When interacting with a SurfaceRadioButton, multiple inputs are accepted. The control state changes from the default state to the pressed state when the first contact touches it. When subsequent contacts are added to the control no additional action is initiated. Release is initiated when the last captured input is been removed from the control while still inside the control. If not previously, the control will be set to 'selected' upon release. It is not possible to transition a SurfaceRadioButton from 'selected' to 'unselected' by touching it.

Benefits & Value

The SurfaceRadioButton provides a physical manifestation of an abstract or metaphorical property, one for which a visual state is not otherwise apparent or appropriate. It provides a focal point for interaction with a clear and well understood metaphor.

Customization

All visual elements of the SurfaceRadioButton can be changed: its shape, its colors, its visual appearance in each of the visual states, and its transition animation can all be customized. It is important that any customization of appearance continues to afford touching by the user. While you may create other designs which are not visually similar to the SurfaceRadioButton in the Surface SDK, it is important that your designs continue to afford touch.

Design Recommendations

- Ensure text label legibility. Be sure that you follow appropriate typography guidelines when typesetting SurfaceRadioButton labels, especially if the object containing the SurfaceRadioButton control(s) can be freely scaled and/or rotated. Use strong sans-serif typefaces, make button labels at least 12 pixels in height, and that the label is legible from all angles and sizes.
- Reserve illumination, effects, animations, or other ornamentation for touch feedback or selected states. SurfaceRadioButton controls should be minimal, understated, and subtle until touched. When touched, a strong visual reaction is appropriate. A somewhat accentuated state is also appropriate for the selected state of a SurfaceRadioButton.
- Volume is good, but avoid directional button movements. Use volumetric shapes and shading to encourage touch and interactions, but do not make SurfaceRadioButton controls appear to

physically depress below the plane of the Surface screen or move in three dimensions. Illuminating from within is a more successful toggle-state or reaction indicator.

Alternative Design Patterns

The wide array of alternative designs made possible by providing alternative visuals for Surface SDK controls means that just about any look and feel can be achieved for SurfaceRadioButton controls. In general, the use of SurfaceRadioButton controls for many situations is generally non-preferred for a Surface experience.

For example, consider the task of creating a shipment of an item between addresses in the user's address book. Three different parameters from the shipment might normally be exclusively selected from distinct lists: the pickup address, the destination address, and the shipping method. In a traditional computer interface, these three might all be selected using SurfaceRadioButton controls. The preferred Surface method, in keeping with our design principles, is a more direct-touch and physical interaction.

A design which better conforms to this is sketched in the next figure. The user is presented with physical representations of the address cards and shipping methods, which are dragged to make up the process for the shipment.

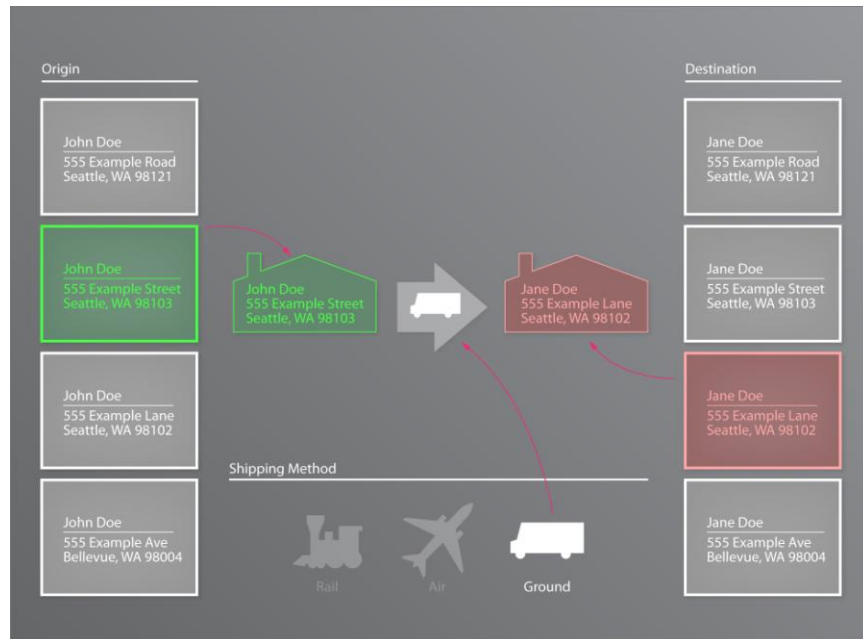


Figure 8.23: An alternative design for selecting shipping origin, destination, and method: the user is asked to drag options to a physical representation of the process.

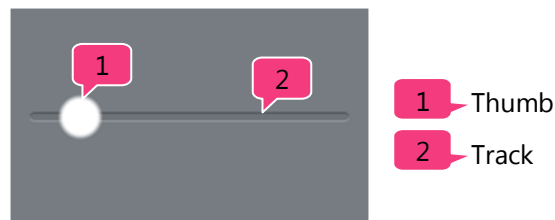
This design can be abstracted to a general pattern of presenting possible choices as physical icons, with the selection made by dragging the appropriate item to a selection area. Another alternative is to keep the options static, with the user asked to drag a 'selected' token to one of the items.

Finally, you should always consider whether a direct physical analog exists for the items being selected. In the case of selecting from among saved addresses, another method is to place the items on a map, and ask the user to select from among them by touching them, or scrolling the map. This is illustrated below.



Figure 8.24: Options are ideally selected from among their physical analogs. In this case, the user selects from saved addresses by selecting the appropriate point on the map. In this design, inspired by the Halo interaction technique, off-screen addresses are shown, along with an arch indicating to the user where the address is located.

8.3.12 SurfaceSlider



Definition

With a SurfaceSlider, users can choose from a continuous range of values. A SurfaceSlider has a bar that shows the range and an indicator that shows the current value. Optional tick marks show values.

Proper Usage

Use a SurfaceSlider when you want your users to be able to set defined, contiguous values (such as volume or brightness) or a range of discrete values (such as levels of difficulty settings).

A SurfaceSlider is a good choice when you know that users think of the value as a relative quantity, not a numeric value. For example, users think about setting their audio volume to low or medium—not about setting the value to 2 or 5.

To decide, consider these questions:

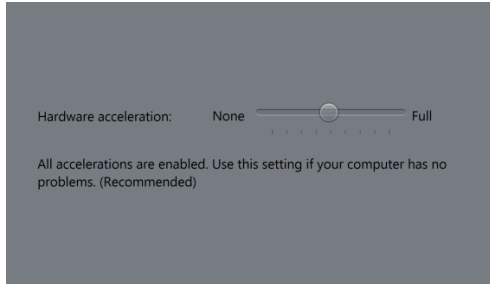
- Does the setting seem like a relative quantity? If not, use SurfaceRadioButton controls or SurfaceListBox controls. Better yet, find an alternative design which does not utilize a traditional control.
- Is the setting an exact, known numeric value? If so, use text boxes.
- Would a user benefit from instant feedback on the effect of setting changes? If so, use a SurfaceSlider. For example, users can choose a color more easily by immediately seeing the effect of changes to hue, saturation, or luminosity values.
- Does the setting have a range of four or more values? If not, use SurfaceRadioButton controls.
- Can the user change the value? SurfaceSlider controls are for user interaction. If a user can't ever change the value, use read-only text boxes instead.

Use a SurfaceSlider if:

- Users will benefit from instant feedback.
- You need to control the range of values which can be input.
- It allows the user to avoid using the on-screen keyboard.

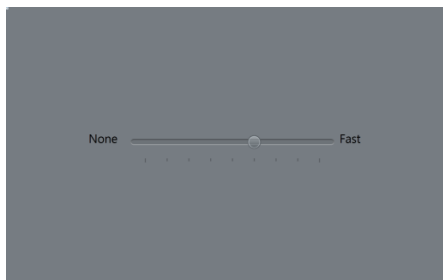
Guidelines

- Use a natural orientation. For example, if the SurfaceSlider represents a real-world value that is normally shown vertically (such as temperature), use a vertical orientation. Because content may be viewed from any angle in a 360 UI, be sure to ground this orientation with adjacent, like-oriented content.
- Orient the SurfaceSlider to reflect the culture of your users. For example, Western cultures read from left to right, so for horizontal SurfaceSlider controls, put the low end of the range on the left and the high end on the right. For cultures that read from right to left, do the opposite.
- Size the control so that a user can easily set the desired value. For settings with discrete values, make sure the user can easily select any value using a finger.
- Consider using a non-linear scale if the range of values is large and users will likely select values at one end of the range. For example, time value might be 1 minute, 1 hour, 1 day, or 1 month.
- Whenever practical, give immediate feedback while or after a user makes a selection. For example, the Microsoft Windows volume control beeps to indicate the resulting audio volume.
- Use tick marks when users need to know the approximate value of the setting.
- Use tick marks and a value label when users need to know the exact value of the setting they choose. Always use a value label if a user needs to know the units to make sense of the setting.
- For horizontally oriented SurfaceSlider controls, place tick marks under the slider. For vertically oriented SurfaceSlider controls, place tick marks to the right for Western cultures; for cultures that read from right to left, do the opposite.
- Place the value label completely under the SurfaceSlider control so that the relationship is clear.

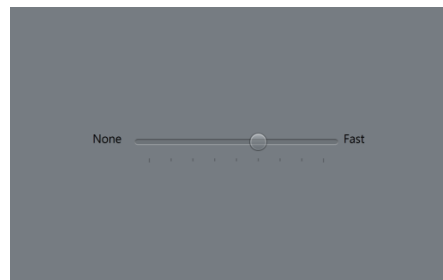


Incorrect: In this example, the value label isn't aligned under the slider.

- When disabling a SurfaceSlider, also disable any associated labels.
- Don't use both a SurfaceSlider and a numeric text box for the same setting. Use only the more appropriate control. Exception: Use both controls when the user needs both immediate feedback and the ability to set an exact numeric value.
- Don't use a SurfaceSlider as a progress indicator.
- Don't label every tick mark.
- Don't change the size of the thumb from the default size.



Incorrect: In this example, a size smaller than the default is used.



Correct: In this example, the default size is used.

Detailed Description

SurfaceSlider controls are intended to select from a range of continuous values. Unlike many other controls, the effect of SurfaceSlider controls is generally carried out while the user is still touching them. This allows for a quick feedback loop, without the need to stop and evaluate between value adjustments.

If enabled, inertia will allow the user to apply a velocity to the thumb, which will carry on after a fast movement until friction brings it to a halt.

There are four states of a SurfaceSlider which can be easily assigned varying looks. In the table below, find a description of each of these states, including a description of the state, as well as the look and feel provided by default in the Surface SDK.

Enabled

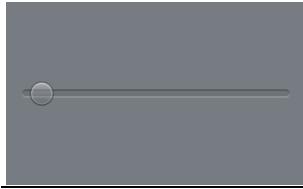
This is the default state for a SurfaceSlider. The user is not in contact with the thumb, but it is available for input.

Disabled

The SurfaceSlider is not available for input. If the user contacts the SurfaceSlider, no action is taken.

- No glow effect on the border

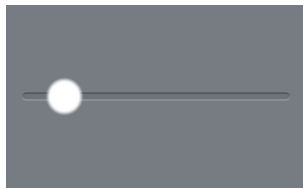
- No glow effect on thumb
- Background gradient is the default color



Captured

The user contacted the Surface on the thumb, and then dragged away from the thumb in a direction non-parallel to the track. The thumb tracks with a change equal to the movement vector of the finger projected onto the line segment of the track. Unlike other controls, the user is able to continue to adjust the SurfaceSlider in this state.

- The thumb glows
- A tool-tip like box can be enabled to show the current value of the SurfaceSlider



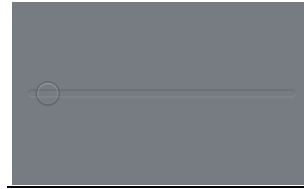
Track Pressed

The user initiated contact on the track area of the SurfaceSlider. Generally, this state is used to cause the value to jump by controlled steps. An alternative mode causes the thumb to jump to the place the user is touching.

- The track grows



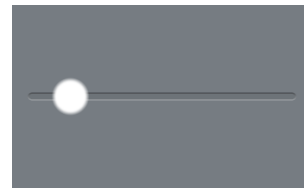
- Thumb and track are lighter (grayed out)



Thumb Pressed

Visibly and functionally identical to the Captured state.

- The thumb grows and turns white



Multi-Touch Behavior

When interacting with a SurfaceSlider, multiple inputs are accepted. SurfaceSlider controls are essentially comprised of two different targets for contacts: the thumb and the track. How each responds to multi-touch input, and how they interact when both are being touched, leads to complicated multi-touch behavior. In particular, it is possible for both the track and thumb to be in their pressed states simultaneously.

Thumb

The thumb transitions from the default to the pressed state when the first contact touches it. When subsequent contacts are added to the thumb no additional action is initiated. If more than one contact has landed on the thumb, then the average movement of those contacts is sent to the SurfaceSlider. Release is initiated when the last captured input is been removed from the control while still inside the control.

If a contact is added to the track after a contact has captured the thumb, the track will then enter the pressed state. This contact will not cause any change in the position of the thumb.

Track

When a contact is added to the track, the track's state changes from the default state to the pressed state. So long as thumb is not in the pressed state, it will to move toward the contact. Each additional contact added to the track will override the previous one, and the thumb will start moving towards the most recently added contact. If the thumb is in the pressed state, however, the contacts on the track will have no influence on the thumb's position.

Benefits & Value

The SurfaceSlider provides a physical manifestation of an abstract or metaphorical property, one for which a physical action is not otherwise apparent or appropriate. It provides a focal point for interaction, with a clear and well understood metaphor.

Even for properties with clear physical manifestations, a SurfaceSlider is an opportunity to disentangle a particular property which might otherwise need to be simultaneously adjusted along with another property. For example, if wishing to precisely rotate an object without moving it: a direct, physical interaction with the object might be inappropriate, since constraining the movement to remove translation will result in a displacement between the object and the fingers. Delegating this to a SurfaceSlider, therefore, maintains the illusion of direct-touch interaction.

Customization

Designers are able to customize the look & feel of this control.

Behavior Customization

The interaction of the thumb and the touch point can be customized. When the user touches the track of the SurfaceSlider, one of three behaviors can be easily configured: repeat, move-to-position, and snap-to-tick:

- **Repeat:** this is the default behavior of the SurfaceSlider: the value jumps by a set increment, such as a page length.
- **Move-to-Position:** the thumb will jump to the position of the contact.

- **Snap-to-Tick:** the thumb will jump to the tick value closest to its current position in the direction of the contact.

Visual Customization

All visual elements of a SurfaceSlider can be changed: its shape, its colors, its visual appearance of each of the states, the appearance and behavior of the track and the thumb, as well as its transition animations can all be customized. A limitation to the customization is that the track must always be a straight line.

It is important that any customization of appearance continues to afford touching by the user. While you may create other designs which are not visually similar to the SurfaceSlider in the Surface SDK, it is important that your designs continue to afford touch.

Design Recommendations

- Design compact, subtle sliders that are easily touchable. Ensure that hit states for thumbs are no less than 38 pixels wide. They should be at least this size when pressed, but may appear visually as small as 7 pixels wide in their enabled (but not pressed) state.
- Design for the grow-upon-touch paradigm. Ensure that sliders and thumbs have larger states that are visible below a user's finger. Slider thumbs could simply scale in size, or have auras around them, but their active states must be visually very clear. Be sure the enlarged state is harmonious with the rest of the slider design and the broader design vocabulary of the experience.

8.3.13 SurfaceScrollBar



Definition

With a SurfaceScrollBar, a user can move to a particular portion of a list, display, or other viewing area. The SurfaceScrollBar is generally reserved for viewport control, with the size of the thumb representing the proportion the current viewport is of the overall viewable region.

Proper Usage

A SurfaceScrollBar is similar to a SurfaceSlider, in that it is designed to select a particular value of a continuous parameter. Unlike a SurfaceSlider, it is reserved for a particular usage scenario: control of a viewport.

Guidelines

- Use SurfaceScrollBar controls only when the content in the list will likely be longer than three times the visible area of the list. If the list's content can be shown in 3 or fewer pages, use the direct manipulation technique of panning the list.
- Don't use SurfaceScrollBar controls on the entire window. SurfaceScrollBar controls are not recommended when contents runs outside of a view area on multiple axes.

Detailed Description

SurfaceScrollBar controls are intended to select the position of a viewport. They differ from SurfaceSlider controls in that the relative size of the viewport is visualized in the size of the scroll bar's thumb. Unlike many other controls, the effect of scroll bars is generally carried out while the user is still touching the thumb. This allows for a quick feedback loop, without the need to stop and evaluate between value adjustments.

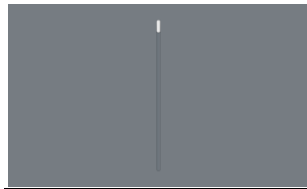
If enabled, inertia will allow the user to apply a velocity to the thumb, which will carry-on after a fast movement until friction brings it to a halt.

There are five states of a SurfaceScrollBar which can be easily assigned varying looks. In the table below, find a of each of these states, including a description of the state, as well as the look and feel provided by default in the Surface SDK.

Enabled

This is the default state for a SurfaceScrollBar. The user is not in contact with the thumb, but it is available for input.

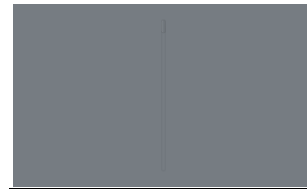
- The thumb is small and within the bar



Disabled

The SurfaceScrollBar is not available for input. If the user contacts the SurfaceScrollBar, no action is taken.

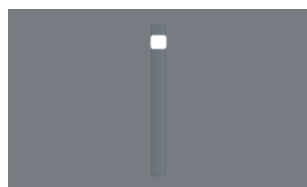
- The track is made lighter (grayed out)
- The thumb is made darker (grayed out)



Captured

The user contacted the Surface on the thumb, and then dragged away from the thumb. The thumb tracks with a change equal to the movement vector of the finger projected onto the line segment of the track. Unlike other controls, the user should be able to continue to adjust the SurfaceScrollBar in this state.

- The thumb and track grow



Thumb Pressed

The user contacted the thumb. Functionally identical to the Captured state.

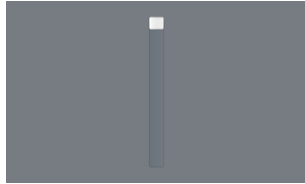
- The thumb and track grow



Track Pressed

The user initiated contact on the track area of the SurfaceScrollBar. Generally, this state is used to cause the thumb to jump to the point of contact.

- The track and thumb grow



Multi-Touch Behavior

When interacting with a SurfaceScrollBar, multiple inputs are accepted. SurfaceScrollBar controls are essentially comprised of two different targets for contacts: the thumb and the track. How each responds to multi-touch input, and how they interact when both are being touched, leads to complicated multi-touch behavior. In particular, it is possible for both the track and thumb to be in their pressed states simultaneously.

Thumb

The SurfaceScrollBar's state changes from the default state to the pressed on thumb state when the first contact touches the thumb. When subsequent contacts are added to the thumb no additional action is initiated. If more than one contact has landed on the thumb, then the average movement of those contacts is sent to the SurfaceScrollBar. Release is initiated when the last captured input is been removed from the control while still inside the control.

If a contact is added to the track after a contact has captured the thumb, the track will then enter the pressed state. This contact will not cause any change in the position of the thumb.

Track

When a contact is added to the track, the track's state changes from the default state to the pressed state. So long as thumb is not in the pressed state, it will to move toward the contact. Each additional contact added to the track will override the previous one, and the thumb will start moving towards the most recently added contact. If the thumb is in the pressed state, however, the contacts on the track will have no influence on its position.

Benefits & Value

Allows the user to have precise control over the current viewport. It also serves as an information display, showing the user how large the overall area is, how large the viewport is as a fraction of that area, and where in the overall area the viewport currently lies.

Customization

Designers are able to customize the look and feel of this control.

Visual Customization

All visual elements of a `SurfaceScrollBar` can be changed: its shape, its colors, its visual appearance of each of the states, the appearance and behavior of the track and the thumb, as well as its transition animations can all be customized.

It is important that any customization of appearance continues to afford touching by the user. While you may create other designs which are not visually similar to the `SurfaceSlider` in the Surface SDK, it is important that your designs continue to afford touch.

Design Recommendations

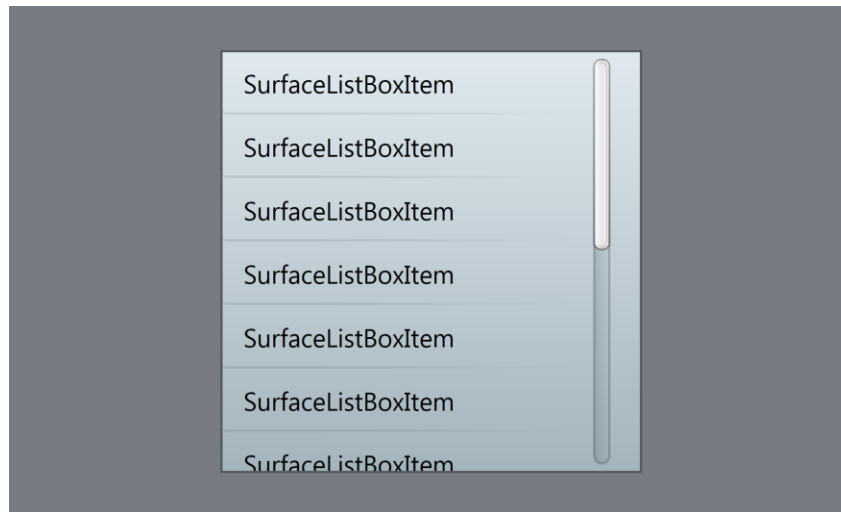
- Design compact, subtle `SurfaceScrollBar` controls while keeping them touchable. Ensure that hit states for thumbs are no less than 38 pixels wide. They should be at least this size when pressed, but may appear visually as small as 7 pixels wide in their enabled (but not pressed) state.
- Ensure enough contrast between the scroll thumb and the scroll track. Since their default sizes should be small, sufficient contrast between the two is vital for the user to recognize that they are `SurfaceScrollBar` controls.
- If paired with a `SurfaceListBox`, let the `SurfaceListBox` take prominence and lead the interaction. Using motion design to lead the eye, having a `SurfaceListBox` animate into place when it instantiates will draw the user's eye to it, and give them strong clues as to how it may be interacted with. In such circumstances, the `SurfaceListBox` should be the user's primary control, and the `SurfaceScrollBar` should be relegated to showing the user's current position within the total list of options.
- Design for the grow-upon-touch paradigm. Ensure that sliders and thumbs have larger states that are visible below a user's finger. `SurfaceScrollBar` thumbs could simply scale in size, or have auras around them, but their active states must be visually very clear. Be sure the enlarged state is harmonious with the rest of the `SurfaceScrollBar` design and the broader design vocabulary of the experience.

Alternative Design Patterns

The wide array of alternative designs made possible by providing alternative visuals for Surface SDK controls means that just about any look and feel can be achieved for `SurfaceScrollBar` controls. In general, the use of `SurfaceScrollBar` controls for many situations is generally non-preferred for a Surface experience.

In many situations, the `SurfaceScrollBar` controls can simply be omitted: rather than requiring users to touch an area offset from the current viewport, we allow users to drag content within the viewport directly to change the view. Such an approach does have three fundamental limitations. First, it omits the information display portion of the `SurfaceScrollBar`, thus depriving the user of a valuable piece of information. Second, it requires users to occlude their content while scrolling. Third, it limits the scope of a particular movement to the size of the viewport. Given the limitations, it is not always an easy choice to omit `SurfaceScrollBar` controls from your system—consider the tradeoffs carefully, and provide design elements to overcome those shortcomings which are important limitations.

8.3.14 SurfaceListBox



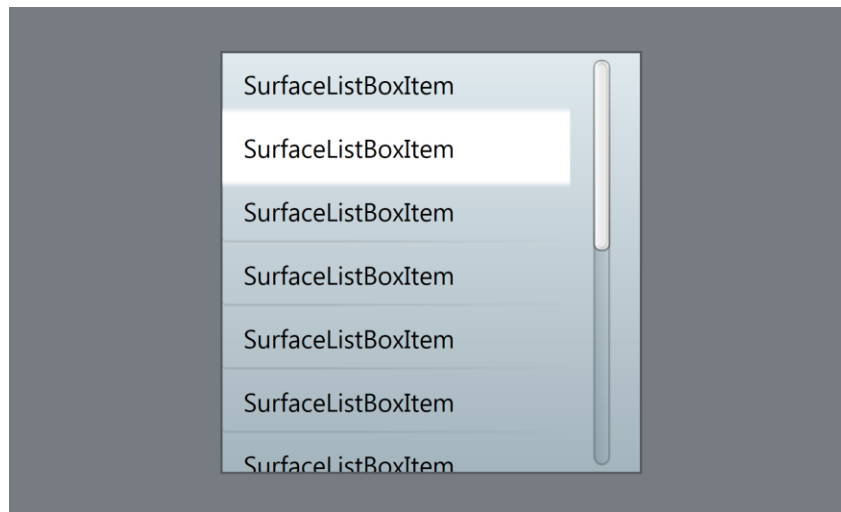
Definition

With a SurfaceListBox, users can select from a set of values presented in a list that is always visible. With a *single-selection* SurfaceListBox, users select one item from a list of mutually exclusive values.

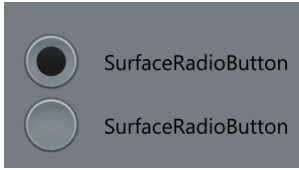
Proper Usage

To decide whether to use a SurfaceListBox, consider these questions:

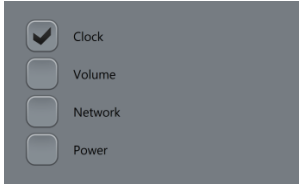
- Is there a lot of content (up to hundreds of items)? Is the content typographic in nature? If so use a *vertical* SurfaceListBox.



- Does the list present data, rather than program options? Either way, a SurfaceListBox is a suitable choice regardless of the number of items. By contrast, SurfaceRadioButton



-
- or SurfaceCheckBox



- are suitable only for a small number of program options.
- Is the content a continuous item, such as a map or large graphic? Use a scroll view.
- Is a 2-dimensional grid of content required? Use a scroll view.

Detailed Description

List boxes are intended to allow the user to select from among multiple items. There are multiple states of a SurfaceListBox which can be easily assigned varying looks. In the table below, find a of each of these states, including a description of the state, as well as the look and feel provided by default in the Surface SDK. Note that, in addition to these states, the varying states of the SurfaceScrollBar portion of the SurfaceListBox (if present) are as described above.

Scrolling in the SurfaceListBox can include inertia: fast scrolling gestures carry on past the release of the contact, with the view eventually coming to a halt with an apparent friction.

The constraints of the SurfaceListBox are conveyed to the user through elasticity: scrolling past the bounds of the list area will eventually be stopped by the control. When the user releases the contacts, the view will 'bounce' back so that only the list is shown.

Enabled

This is the default state for a SurfaceListBox. The user is not in contact with the list, but it is available for input.

- The thumb is small and within the bar

Disabled

The SurfaceListBox is not available for input. If the user contacts the SurfaceListBox, no action is taken. Note that it is also possible to disable individual items in the SurfaceListBox.

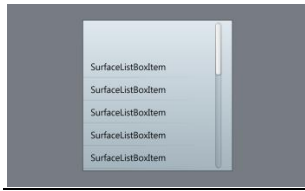
- The text are made lighter (grayed out)
- If the entire SurfaceListBox is disabled, the SurfaceScrollBar is also made lighter (grayed out)



Captured

The user contacted the Surface on SurfaceListBox, and then dragged away from the SurfaceListBox. The list tracks with a change equal to the movement of the finger projected onto the track. Unlike many other controls, the user is able to continue to adjust the SurfaceListBox in this state.

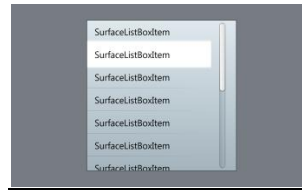
- The list scrolls with the finger



Item Pressed

When any item in the list is pressed the item under the contact immediately enters its pressed state. Movement of the contact then determines the associated behavior. If the contact moves up or down, the list will immediately start scrolling.

- Item under contact becomes 'pressed'



Item Selected

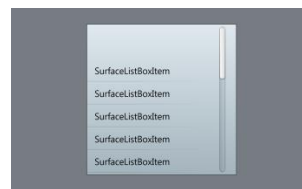
When the user has pressed on an item and removed their finger, that item is 'selected'.

- Item's background color is adjusted to signify selection



Scrolling

Once the list enters scroll mode the list's appearance fades to its drag state within 0.2 seconds. While scrolling the list should track as closely as possible to the movement of the content. Do not add physics that add lag to the interaction. If the contact moves fewer than 15 pixels and releases the content under the contact becomes selected. When all the contacts are released and there is still vertical movement, the list should continue at the speed it was moving when the contacts were released. Then the list should constantly slow to a halt. Adjust the amount of "friction" in order to make the list feel natural and intuitive.



Multi-Touch Behavior

When interacting with a SurfaceListBox, multiple inputs are accepted. A SurfaceListBox is essentially comprised of two separate controls: a list of items, and a SurfaceScrollBar. For details on multi-touch behavior of the SurfaceScrollBar, see above.

When a single contact is added to the SurfaceListBox the view starts tracking 1:1 with any movement of the contact. In a SurfaceListBox the movement is constrained to one axis so the SurfaceScrollViewer inside the SurfaceListBox will track 1:1 with the contact within that constraint. When an additional contact is

added to it, the average movement of the two contacts will be transferred to the list. The same applies to more than 2 contacts; the overall movement is averaged and sent to the SurfaceListBox.

Benefits & Value

A SurfaceListBox provides a convenient mechanism for browsing and selecting from among a list of items.

Customization

- The default SurfaceListBox control is mostly transparent, so place it on a light background so the dark text will stand out.
- You can change the background of the SurfaceListBox and still retain the divider lines and select state. When changing the background keep a light value unless you also plan to change the text color, selection and pressed states.

Design Recommendations

- Ensure text label legibility. Be sure that you follow appropriate typography guidelines when typesetting list box content, especially if the object containing the list box control(s) can be freely scaled and/or rotated (see [Typography](#)). Use strong sans-serif typefaces, make list item text at least 12 pixels in height, and that the text is legible from all angles and sizes. Flat or minimally shaded backgrounds behind each SurfaceListBoxItem will help with legibility.
- Enable passive learning through animation to show how list boxes can be used. Animating a list box's contents into place when it instantiates will draw the user's eye to it, and give them strong clues as to how it may be interacted with. Having seen some content scroll out of view, flicking the SurfaceListBox to view that content is an easy inference to draw for most users.
- If paired with a SurfaceScrollBar, let the SurfaceListBox take prominence and lead the interaction. A SurfaceListBox should be the user's primary control, and the SurfaceScrollBar should be relegated to showing the user's current position within the total list of options.
- Design for the grow-upon-touch paradigm. Ensure that scroll bars and thumbs have larger states that are visible below a user's finger. Scroll bar thumbs could simply scale in size, or have auras around them, but their active states must be visually very clear. Be sure the enlarged state is harmonious with the rest of the scroll bar design and the broader design vocabulary of the experience.
- Reserve illumination, effects, animations, or other ornamentation for touch feedback or selected states. SurfaceListBox contents should have a strong visual reaction when touched. A somewhat accentuated state is also appropriate for the selected state of a SurfaceRadioButton.

Alternative Design Patterns

Many of the same design patterns described for SurfaceCheckBox controls and SurfaceRadioButton controls apply to list boxes.

Examples

An example of the preferred usage of a SurfaceListBox is in the music player described previously. In this case, the selection and playing of a song happen in one step, rather than two as might otherwise be the case.

The album's song list in the Music application is contained within a SurfaceScrollViewer. In this case,

scrolling is constrained to the vertical, and SurfaceScrollBar controls are again omitted. In this application, touches to the song cause it to play, but grabbing a song and dragging it vertically causes the content to scroll.

A small 'cheat' is included in this application: when the album flips over from the artwork to the track list, the ScatterViewItem grows slightly in the vertical to allow more tracks to be shown. This is done to reduce the amount of scrolling needed to interact with the content.



Figure 8.25: The song list in the Music application is contained within a SurfaceListBox. Dragging vertically causes the view to scroll.

8.3.15 SurfaceTextBox



Definition

SurfaceTextBox is a typical text box, a place where a user can enter text. Whenever the user touches a SurfaceTextBox, the on-screen keyboard will appear in order to allow the user to enter text. This keyboard takes on two forms (above): a regular alphanumeric keyboard, and a numeric keypad, which offers only numeric input.

Proper Usage

To decide if this is the right control to use, consider these questions:

- Is the valid data completely unconstrained, constrained only by format (constrained length or character types)? If so, use a `SurfaceTextBox`.
- For numeric data:
 - Does the user perceive the setting as a relative quantity? If so, use `SurfaceSlider`.
 - Would the user benefit from instant feedback based on what he or she is entering? If so, use a `SurfaceSlider`, possibly along with a `SurfaceTextBox`. For example, a user can easily choose a color using a `SurfaceSlider` because he or she can immediately see the effect of changes to hue, saturation, or luminosity values.

While a `SurfaceTextBox` has the benefit of being very flexible, it has the drawback of having minimal constraints. The only constraints on an editable `SurfaceTextBox` are:

- You can choose to set the maximum number of characters.
- You can choose to restrict input to numeric values (0 – 9) only.

Aside from its length, `SurfaceTextBox` does not have any visual clues that suggest the valid values for the data or its format. This means you must rely on labels to convey this information to the user. If the user enters text that is not valid, you must handle the error with an error message.

As a general rule, you should use the most constrained control that you can. Use unconstrained controls like `SurfaceTextBox` as a last resort. Yet be careful as well, when you are considering constraints, to bear in mind the needs of global users. For example, a control that is constrained to U.S. ZIP codes is not globalized, therefore an unconstrained `SurfaceTextBox` that accepts any postal code format would be preferred.

Detailed Description

When a user touches a `SurfaceTextBox`, it is given focus. At that time, the on-screen keyboard is invoked, which the user can touch in order to enter text into the system. Only one on-screen keyboard can be visible at a time. The type of on-screen keyboard that appears (alphanumeric keyboard or numeric keypad) depends on the type of `SurfaceTextBox`. If the user is entering values into a numeric `SurfaceTextBox`, then the numeric keypad appears. Otherwise, the regular alphanumeric keyboard is shown.

Enabled

This is the default state for a `SurfaceTextBox`. The user is not in contact with the `SurfaceTextBox`, but it is available for input.

- The background is dark

Disabled

The `SurfaceTextBox` is not available for input. If the user contacts the `SurfaceTextBox`, no action is taken.

- The text is made lighter (grayed out)
- The background is made lighter (grayed out)



Captured

The user touches the area outside of the SurfaceTextBox, then drags his or her finger over the SurfaceTextBox or onto the SurfaceTextBox, and then drags his or her finger off. Generally this state should not be used to initiate actions with a SurfaceTextBox.

- Visually matches the 'pressed' state



Pressed

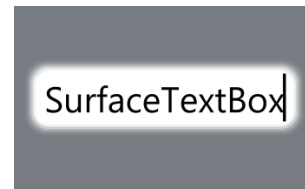
The user touches the SurfaceTextBox. At that moment, the on-screen keyboard appears (if not already present) to allow the user to enter text. Generally this state should not be used to initiate any actions.

- Background becomes lighter
- Cursor appears for text entry
- A glow effect is added around the SurfaceTextBox
- The on-screen keyboard appears



Focused

The user touches the SurfaceTextBox, and then removes his or her finger from the screen. This state is functionally identical to the Pressed state.



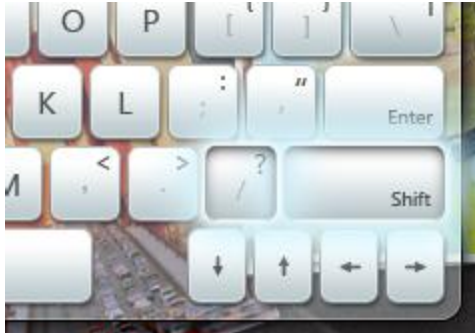
Text Selected

The user drags his or her finger over a portion of the text within a SurfaceTextBox and that text becomes selected.

- Background becomes lighter
- Cursor is hidden
- A highlight appears around the selected portion of text
- Pressing a key on the on-screen keyboard immediately deletes the selected text and replaces it with the character shown on that key

Multi-Touch Behavior

When the user interacts with a SurfaceTextBox, only a single input is accepted. The most recent touch is taken to be the currently active contact. When the user interacts with the on-screen keyboard, however, multiple inputs are accepted. Each input is treated as a separate touch, and multiple keys can be depressed simultaneously, similar to a physical keyboard.



Guidelines

General

- All SurfaceTextBox controls
 - When you are disabling a SurfaceTextBox, also disable any associated labels, instruction labels, and SurfaceButton controls.
 - Use auto-complete to help users enter data that is likely to be used repeatedly. Some examples include user names, addresses, and file names. However, do not use auto-complete for a SurfaceTextBox that may contain sensitive information, such as passwords, PINs, credit card numbers, or medical information.
 - Do not make users scroll unnecessarily. If you expect data to be larger than the SurfaceTextBox and you can readily make the SurfaceTextBox larger without harming the layout, resize the box to eliminate the need for scrolling.



Incorrect: In this example, the SurfaceTextBox should be made much longer to handle the data.

- Using a SurfaceScrollBar with a SurfaceTextBox:
 - Do not put a horizontal SurfaceScrollBar on a multi-line SurfaceTextBox. Use a vertical SurfaceScrollBar and line wrapping instead.
 - Do not put a SurfaceScrollBar on a single-line SurfaceTextBox.
- Editable SurfaceTextBox controls
 - Limit the length of the input text field when you can. For example, if the valid input is a number between 0 and 999, use a numeric SurfaceTextBox that is limited to three characters. All parts of a SurfaceTextBox that use formatted data input should have a short, fixed length.
 - Be flexible with the data formats that you accept. If the user is likely to enter text using a wide variety of formats, try to handle all of the most common ones. For example, many names, numbers, and identifiers can be entered with optional spaces and punctuation, and the capitalization often does not matter. You should be prepared to accept all these variations.

- If you cannot handle a wide variety of formats, clarify the specific format by using formatted data input (see below) or indicate the valid formats in the label.

Acceptable: In this example, a SurfaceTextBox requires input in a specific format.

Better: In this example, formatted data input is used to require a specific format.

Best: In this example, a SurfaceTextBox handles a variety of formats.

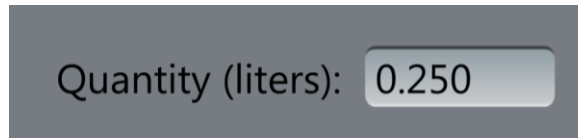
- Consider format flexibility when choosing the maximum input length. For example, a valid credit card number can use up to 19 characters; therefore it would not be wise to limit the input length to anything shorter.
- Do not use formatted data input if the user is more likely to paste in long, complex data. Rather, reserve formatted data input for situations where the user is more likely to type the data.

- In this example, the formatted data input pattern is not used, so that the user can directly paste in his or her IPv6 address.
- If the user is more likely going to reenter the entire value, select all the text on input focus. If the user is more likely to edit, place the cursor at the end of the text.

- In this example, the user is more likely to replace the text rather than edit it; therefore the entire text field is selected on input focus.

- In this example, the user is more likely to add keywords rather than replace the existing text; therefore the cursor is placed at the end of the text on input focus.
- Always use a multi-line SurfaceTextBox if new-line characters are valid input.
- When the SurfaceTextBox is being used to acquire a file or path, always provide a Browse button.

- In general, whenever possible, provide the user with the option of entering a common value other than by typing it.
- Numeric SurfaceTextBox controls
 - Choose the most convenient unit and label the units. For example, consider using milliliters instead of liters (or vice versa), percentages instead of direct values (or vice versa).

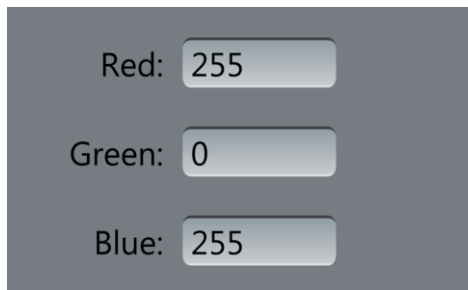


Acceptable: The unit is labeled, but it requires the user to enter decimal numbers.



Better: The SurfaceTextBox uses a more convenient unit.

- Right-align numeric values whenever the following situations occur:
 - There is more than one numeric SurfaceTextBox.
 - The numeric text fields are vertically aligned.
 - The user is likely to add or compare the numeric values he or she enters.
 - Always right-align monetary values.

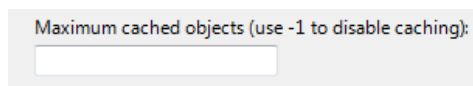


Incorrect: In this example, the numeric text is incorrectly left-aligned.

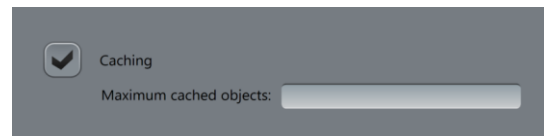


Correct: In this example, the numeric text is right-aligned to make it easy to compare values.

- Do not assign special meanings to specific numeric values, even if those special meanings are used internally by your application. Instead, use SurfaceCheckBox or SurfaceRadioButton for an explicit user selection.



Incorrect: In this example, the value -1 has a special meaning.



Correct: In this example, a SurfaceCheckBox makes the option explicit.

Password and PIN input

- Always use the SurfacePasswordBox (see next section) instead of creating your own secure version of the SurfaceTextBox. Passwords and PINs require special treatment in order to be handled securely.

Textual output

- Consider using the white background system color for large, multi-line read-only text. A white background makes the text easier for the user to read. Large quantities of text on a gray background can discourage the user from reading.

Data output

- Do not use a border for a single-line, read-only SurfaceTextBox. The border is a visual clue that the text is editable.
- Do not set a tab stop on a single-line, read-only SurfaceTextBox unless the user is likely to need to scroll or copy the text.

Input validation and error handling

Because a SurfaceTextBox is usually not constrained to accept only valid input, you may need to validate the input and handle any problems yourself. Validate input problems as follows:

- If the input data is invalid, wait to give an error message only when the entire input is complete and the user touches an on-screen SurfaceButton to indicate completion. Do not notify the user while he or she is typing, instead wait until completion.

Prompts

A prompt is a label or short instruction placed inside a SurfaceTextBox as its default value. Unlike static text, prompts disappear from the screen once the SurfaceTextBox receives input focus.



A typical prompt: "Search" will disappear when the focus is on that particular SurfaceTextBox.

- Use a prompt when screen space is at such a premium that using a label or instruction is undesirable, such as on a toolbar.
- The prompt is primarily for identifying the purpose of the SurfaceTextBox in a compact way. It must not contain crucial information that the user needs to see while using the SurfaceTextBox.
- Do not use prompts for directing the user to type something specific or to click a specific buttons. For example, do not write prompts that say, "Enter a filename and then click Send."
- The prompt text must not be confused with input text, so be sure to keep it visibly distinctive.

When using prompts:

- Draw the prompt text in italic gray and the actual input text in normal black.
- Keep the prompt text concise. You can use fragments instead of full sentences.

- Do not use ending punctuation or ellipsis.
- The prompt text should not be editable and should disappear once the user clicks in or tabs into the SurfaceTextBox.
- Exception: If the SurfaceTextBox has default input focus, the prompt is displayed until the user starts typing.
- Restore the prompt text if the SurfaceTextBox is still empty when it loses input focus.

Benefits and Value

A SurfaceTextBox provides an unconstrained data entry field. The SurfaceTextBox control provided in the Surface SDK also gives access to the on-screen keyboard, without any need for the developer to specifically invoke the keyboard.

Customization

All visual elements of the SurfaceTextBox can be changed: its shape, its colors, its visual appearance in each of the visual states, and its transition animation.

The on-screen keyboard cannot be customized at this time. It appears either as an alphanumeric keyboard or a numeric keypad, as shown previously. It is not a regular control, and it cannot be visibly embedded into your application or into other controls. Its appearance behavior is fixed and it will only appear when the user touches a SurfaceTextBox to enter text.

Design Recommendations

- Do not alter the default behavior for the on-screen keyboard. A great deal of research has led to the default behavior of the on-screen keyboard. Its initial position with respect to the destination SurfaceTextBox, and the movement of the on-screen keyboard when a user changes to another SurfaceTextBox, are all tightly controlled by default, and should not be changed.
- Be smart about control orientation. Give careful consideration to a SurfaceTextBox's surrounding objects or parent control, and its orientation. If a SurfaceTextBox's parent object can be rotated, the SurfaceTextBox will need to be larger and bolder to retain legibility at all times.
- Ensure text label legibility. Be sure that you follow appropriate typography guidelines when typesetting a SurfaceTextBox, especially if the object containing the SurfaceTextBox can be freely scaled and/or rotated). Use strong sans-serif typefaces, make text at least 12 pixels in height, and ensure that the label is legible from all angles and sizes.
- Maintain a visual rhythm if you need more than one SurfaceTextBox. Unify the justification, width, and spacing of any related SurfaceTextBox controls if they must be shown on-screen at once. Look for opportunities to simplify the screen by waiting until later to have another SurfaceTextBox, if appropriate.
- Think beyond the on-screen keyboard. Consider how else text might be entered into the system rather than the on-screen keyboard, such as object recognition.

Alternate Design Patterns

Although the on-screen keyboard has been thoroughly designed and tested, it is still time consuming to enter text by typing. It is generally preferred that typing be avoided in Surface applications. Whenever possible, replace text entry fields with physical manipulation metaphors. For example, when entering an address, allow the user to select the components of his or her address from among choices on a list (such

as state) or try to automatically generate those components which are a function of the rest of the content (such as ZIP code).

8.3.16 SurfacePasswordBox



Definition

A SurfacePasswordBox is a text box that allows a user to enter a piece of confidential authentication information, such as a password. Unlike a regular SurfaceTextBox, the characters do not appear in the box, enhancing security. The overall behavior of the SurfacePasswordBox is otherwise the same as the SurfaceTextBox previously described.

Security Considerations

The physical characteristics of Surface make it difficult for a user to hide his or her fingers while entering passwords. To help protect the user's privacy, the visual feedback is removed from the on-screen keyboard while it is being used to enter text into a SurfacePasswordBox, and the keys do not highlight.



Figure 8.26: Left: The visual feedback that appears when a user presses 'T' while entering data for a SurfaceTextBox is shown. Right: When a user enters data for a SurfacePasswordBox, the visual feedback is removed.

9.0 Methods and Tools



The quality and immersive nature of Microsoft Surface experiences are essential to Surface applications. The goal of these applications is to achieve a level of emotional bonding that is not achievable with traditional GUI applications, either in Windows or on the Web. Emotionally bonding user experiences are becoming necessary to capture the mindshare—and the pocket share—of today’s increasingly discerning consumers. To achieve this quality of user experience, it is important to rely on design methods outlined in this section. Many of the traditional design methods continue to apply, and some methods need to be adjusted. Some additional methods are strongly recommended, such as bringing users into the design process earlier than with a traditional GUI application, engaging more deeply in participatory design, and fine-tuning the experience before reaching the validation stage that typically focuses on usability and rather constrained adjustments.

9.1 Research

9.1.1 *Evaluating a Microsoft Surface Experience*

The purpose of any evaluation is either explicit or implicit. The more explicit the purpose, the greater the likelihood that the study will be successful and the findings will be clearly understood. There are three categories of purposes for a Surface experience evaluation:

1. **Exploratory research** identifies new problems.
2. **Generative research** identifies new product opportunities by providing insight into people’s unmet needs.
3. **Evaluative research** tests the feasibility of a solution by using factual evidence and data.

9.1.2 *Exploratory Research*

Exploratory research is conducted because a problem has not been clearly defined, usually at the beginning of the project or design iteration. The result of exploratory research is a problem statement and/or insights.

Discovering Usage of Reaction

Exploratory research methods study interaction in a specific context with a minimal set of preconceptions and an explicit goal of uncovering unforeseen aspects of the product. The best known of these methods is Contextual Inquiry, where users are interviewed while they work. Another common method is diary studies, where users keep diaries of their usage. Discovery methods also include the empirical logging of usage of a system in the context of its deployment. Although these approaches differ from each other, they share a common goal: to discover unanticipated results from usage of and reaction to the deployment of technology.

9.1.3 Generative Research

Generative research is often conducted during the incubation stage of product development when the product is still a nascent idea. It can also be conducted during the design phase to inform iterations and improvements to design solutions based on data and user participation.

Improving an Application or Design

The purpose of this type of evaluation is simple, although often misunderstood. Its goal is to improve a component, an interface, or an entire system by providing feedback on a prototype at any stage in the design phase. Methods include typical usability studies, RITE (Rapid Iterative Testing Evaluation) studies, and other non-test methods such as Heuristic Reviews and Cognitive Walkthroughs.

Determining Configuration

This type of evaluation takes existing elements of a non-working application and combines them like building blocks to produce a working application. Participatory design, where the users are directly involved in the early stages of design, serves as a primary methodology for configuration. The users are considered partners in the design process, whereas in Contextual Inquiry, they are partners in the research process.

9.1.4 Evaluative Research

Evaluative research tests the feasibility of a solution by using factual evidence, data, and user participation. This research can occur during the design phase to evaluate solutions to components, interfaces, or an entire application. It can also occur after code completion.

Deciding between Alternatives

Most classical scientific experiments fall into this category where data is collected to decide between alternatives. In some cases, the alternatives serve as expressions of more general theories, e.g., do objects with greater or lesser mass fall faster? In other cases, they are merely practical, e.g., does this drug treat heart disease better than a placebo? A simple example in Surface is testing readability in order to decide between different fonts.

Setting Parameters

Any interactive system where user input and system response are continuous requires the definition of parameters that describe the relationship between user input and system response. Examples include:

- The relationship between the degree of deflection of a joystick (user input) and the rate of movement of a targeting reticle (system response)
- The relative movement of a cursor in relation to the movement of a mouse
- The relative movement of a finger on a touch pad (user input) in relation to the relative movement of a cursor (system response)

At times these relationships may be quite complex and context sensitive, such as the rate of movement of an object when you move your finger across a Surface application. The goal is to tune the parameters of the experience until they feel correct.

Scoring a Design or Application

A design is typically evaluated when it reaches a certain level of completion, and it is ready to be

compared with other evaluations performed earlier. These evaluations are commonly called benchmark tests. They serve both to evaluate the current state of the design and to set the empirical user experience metrics for relevant tasks.

9.1.5 Implications to the Research Flow

Designing for Surface experiences requires close collaboration with design and research so that the research and user participation can inform solutions throughout the design phase.

The Old Way:

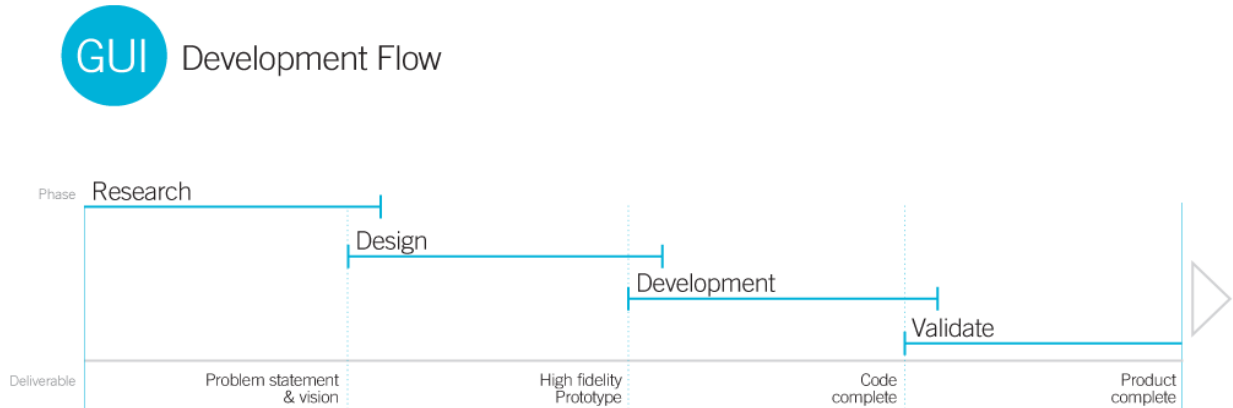


Figure 9.1: Waterfall developments restrict when research happens throughout a design and development project.

The problem with the GUI development flow is that research usually ends before design begins, and concept value testing during the design phase is not always observed. The user provides initial insights before the design phase begins, and usually does not see the product again until it is already in development. Due to the physical nature of the designs, constant observation by users is necessary in order to achieve compelling results.

The New Way:



Figure 9.2: More iterative design methods allow for constant, iterative research and validation of all concepts, ideas, and designs.

In NUI development, user participation (not just feedback) is necessary throughout the design process so that users can inform the experience throughout the design phase. Accommodations for research must be made in the design phase so that it overlaps the research phase. In addition, constant evaluation of design iterations should take place during the design phase to help inform improvements. These changes can help promote a sense of playful interaction that is part of the Surface philosophy. As a result, the research phase continues well into the design phase, and the entire organization accommodates the following:

Exploratory research ends earlier. This type of research results in initial insights and problem statements. It initiates the design phase and Generative research that allows user participation to inform the design iteratively throughout this phase.

- **Generative research and design begin at the same time.** The research and design phases blend together due to the need to get the user's perspective continuously during the design process.
- **Evaluation and generative research are done at the same time.** Constant evaluation of the many design iterations should take place during the design phase to help inform improvements to the solutions.
- **Product evaluation is done in less time.** Because time is needed for rapid prototyping and iterative development, code completion will likely be achieved later than it typically would in GUI development. However, the need for final validation and tweaks should be reduced due to the availability of user input throughout the process.

9.1.6 Research Methods

This section briefly describes the methodology used in each type of research. It is not intended to be complete or exhaustive. The discipline of User Research is continuously developing new methods that are often adaptations of methods applied in other domains or new approaches that are hybrids of many existing research methods. The following list of methodologies is recommended when designing for Surface experiences. These methodologies provide a better understanding of users needs and aspirations, and help support the iterative design process for Surface experiences.

There are two types of research data that can be collected by using many methods:

- **Quantitative research** is a numerical representation of data and information, and meaning can then be calculated from the data.
- **Qualitative research** is a descriptive form of data and information that is communicated through mechanisms such as words, interviews (audio,) pictures, video, and artifacts.

	Purpose	Method(s)
Exploratory	Discovering usage/reaction	Technology probes Contextual inquiry Observation In-depth interviews Precision questioning
	Improving app/design	Usability studies Rapid Iterative Testing Evaluation Heuristic reviews Cognitive walkthroughs Thinking aloud Forced choice observation
Generative	Determining configuration	Participatory design Conjoint testing
	Deciding between alternatives	Comparative experiments
Evaluative	Setting parameters	Testing many prototypes Adjusting prototypes on the fly
	Scoring an app/design	Benchmark studies Metrics for desirability

Figure 9.3: There are many available methods for every type of research, making it easy to choose the right method for the right project.

Technology Probe

This method uses basic and primitive prototypes that are created independently from the technology so

that the focus is on the potential behaviors instead of technical aspects. The prototypes are sent to participants who catalog the ways they use them and integrate them into their lives. The result is an understanding of the behaviors and affordances in the context of the participants' environments.

Contextual Inquiry

In contextual inquiry, participants are interviewed while they work at their own work-sites in order to find the motivations for the activities they perform. A common tool used by the participants is a diary in which they document their usage. Researchers and designers of the Concierge application used contextual inquiry to ask hotel concierges about the kinds of requests they received most often, the recommendations that were the most popular, and the manner in which they directed guests to their destinations.

Ethnographic Research

There are many methods that fall into the category of ethnographic research. These methods result in insights, problem statements, and recommendations about a design solution. Following are just a few of the tools that ethnographic research employs:

- **Observing.** This can be done in the participant's home or the context in which he or she will be interacting with a system. The goal is to observe the behaviors of a user within the intended environment. For example, researchers and designers of the Concierge application observed hotel guests who were looking through brochures in hotel lobbies to find local activities and entertainment.
- **In-depth interviews.** These are normally conducted in person, and are used to guide the participant through a set of scenarios and tasks in order to understand his or her behaviors and needs.

Usability Study

Usability studies are used to evaluate and understand a solution to a component, an interface, or an entire application. A simple scenario-based walkthrough or full working prototype can be used to take participants through a set of tasks and observe the way they use the solution.

Rapid Iterative Testing Evaluation (RITE)

The Rapid Iterative Test and Evaluation (RITE) method is a highly flexible and effective method based on the following goals and assumptions, and is most effective with low-fidelity prototypes that can be quickly changed between scheduled sessions:

- Its primary goal is to improve products as rapidly and efficiently as possible. Conversely, the goal of RITE is NOT to uncover all usability issues.
- Its secondary goal is to "evangelize" user research and iterative testing with design and development teams. To do this, any method must show quick and effective results. RITE empowers teams to fix bugs quickly.
- A primary assumption is that a cross-discipline team of user researchers, designers, and developers is best able to determine the seriousness of the issue for users, the generalizability (how likely a user is to experience this issue), and the possibility of fixing the issue. On this basis, the team can "triage" user issues and fixes.
- A secondary assumption is that it is better to quickly verify that fixes actually work for users (and alter the fixes if they do not) than to repeat testing of the same interface to "validate" an issue or

assess its importance.

This filtering of problems using the aforementioned goals and assumptions is based on a shared understanding of the users, their tasks, the purpose of the application, and the development environment. Such a shared understanding capitalizes on the team's intelligence and judgment. It contrasts sharply against some traditional approaches to usability testing that focus on counting the number, duration, or proportion of problems to determine the importance or generalizability of a problem and the effectiveness of a solution.

Since the RITE method presents a broad framework for understanding users and improving products, it has a number of variations and it offers a "platform" for the creativity of a team.

Heuristic Reviews

This method is recommended throughout the Surface design phase to evaluate whether solutions are adhering to the design principles. Experts review solutions and indicate where the experience and principles diverge.

Cognitive Walkthroughs

Since Surface experiences aim to solve interaction problems in new ways, cognitive walkthroughs are especially useful. Based on the user's goal, the method takes a participant step-by-step through a set of tasks and evaluates the ease or difficulty in completing each task. This method is best applied throughout the entire design phase to evaluate solutions and make sure that the goal is easily achievable.

Thinking Aloud

This method requires that participants speak their thoughts aloud as they try to complete a task or interaction. The outcome provides a better understanding of their behaviors, aspirations, and needs for each task or interaction. When designing Surface experiences, thinking aloud helps to understand the way that a user attempts to complete a task or achieve a goal.

Participatory Design

The participatory design method is used when a configuration of an interface or application is not defined, for example, when components are available but not in order. Participants are asked to use various tools and techniques to organize the components or define the order in which they appear.

Comparative Experiments

This method is used to decide between alternate solutions. The method can be employed by developing a variety of experiments, visualizations, or prototypes that can then be compared. For example, designers of Surface experiences can use this method to compare various fonts or touch sizes in the interface.

Benchmark Studies

Benchmark studies are most valuable when they are performed throughout the Surface development flow and in the validation phase, although they can be time-consuming and expensive. Benchmark studies analyze performance by repeatedly testing core tasks in the application against a pre-measured quality or goal to determine overall performance and usability.

Desirability Toolkit

Surface designers must create experiences that are desirable. The desirability toolkit is a simple method that uses card-sorting techniques to identify relevant and irrelevant qualities of an experience. Each card

has a single word or image. Participants are asked to select the cards that are most relevant to the experiences. This is an inexpensive, efficient method that helps evaluate the qualitative and emotional component of the design. Combined with traditional usability techniques, the aim of using the toolkit is to understand the intangible benefits gained from using the product.

Pre-Verbal User Response Evaluation (PURE)

Surface designers must create experiences that go beyond the functional and useful. To do this successfully, designers face two challenges:

- How do we empirically assess the level of emotional engagement users have with the products we create?
- How do we uncover the specific design elements and attributes in a product that add to or detract from a deeper, more desirable, and engaging product experience?

This method directly addresses the challenge of measuring the quality of experiences. The basic theory behind PURE is the James-Lange Theory of Emotion. This cognitive theory suggests that emotions occur as a result of physiological reactions to events. According to this theory, if users experience an external stimulus that leads to a physiological reaction, their emotional reactions are based on how they interpret those physical reactions. The PURE methodology uses these physiological reactions combined with magnitude estimation to enable a measurement of the desirable and undesirable qualities of an experience.

End-State

This method provides the greatest value when testing the success of new gestures designed for Surface experiences. It tests the effectiveness of gestures by providing a starting state and an ending state, and asking users to demonstrate the most effective gesture to move from one state to the other.

9.1.7 Research Tools and Techniques

Following are a few of the many tools and techniques that can be used to perform the research methodologies described earlier.

	Purpose	Method(s)	Tools & Techniques
Exploratory	Discovering usage/reaction	Technology probes Contextual inquiry Observation In-depth interviews Precision questioning	Affinities Design action table Logging data Statistical and conditional probability reports Quick prototypes Camera/video/audio recording
	Improving app/design	Usability studies Rapid Iterative Testing Evaluation Heuristic reviews Cognitive walkthroughs Thinking aloud Forced choice observation	Scenario walkthroughs Working prototypes Metrics Post-test questionnaires
Generative	Determining configuration	Participatory design Conjoint testing	Collaging Scrapbook building User-created scenarios Thinking aloud Pre-grouped combination prototyping
	Deciding between alternatives	Comparative experiments	Experimental designs Statistical evaluation of results Pre-grouped combinations
Evaluative	Setting parameters	Testing many prototypes Adjusting prototypes on the fly	High-fidelity prototypes capable of adjusting quickly Regression analysis
	Scoring an app/design	Benchmark studies Metrics for desirability	Standard set of questions and process Scorecards

Figure 9.4: There are many effective methods available for any type of research.

Affinity

Sometimes referred to as the KJ Method by Jiro Kawakita in the 1960s, affinities group ideas and data for analysis. The primary goal of an affinity is for the participating team to reach a consensus. This team consists of key stakeholders in the product's design, and allows them to voice their input and have their points of view addressed. An affinity also highlights commonalities as well as gaps in information that may require further research before any decisions are made. The commonalities that emerge from the tool are the result of an organic process that is not guided by any one bias, and represent a holistic point of view. The outcome is grounded by actual evidence from research rather than by any preconceived notions of the solution.

Data Action Table

A data action table is a method used to capture design changes that the research team recommends to the design team based on user data and feedback. It includes the recommendation, the data source, and the benefit that will result from the design change. The changes are then prioritized for the design team. Following are two examples:

FEATURE/TASK	P	ACTION	DATA	BENEFIT
User Interface Design	1	Move the advanced search link closer to simple search area.	User could not find “advanced search” link (had to search in sitemap).	The user is more likely to use the advanced search feature if it's easier to find.
Advanced Search Feature	1	Attempt to incorporate the advanced search filters into the simple search feature.	The user indicated she preferred the simple search feature, “I’m lazy. I like the quick search.”	The advanced search filters become more available (and relatable) in a feature the user already interacts with.

Logging Data

Keeping a paper trail of information requires a large upfront commitment in any research endeavor, but it will pay off in the end. Logging data helps eliminate any ambiguity in the insights drawn from research by recording the context in which they were uncovered. Additionally, a log kept in a digital format can be aggregated into a more meaningful collection. Tools such as Microsoft Office Excel® spreadsheet software and SPSS statistical software enable practitioners to divide data into meaningful chunks.

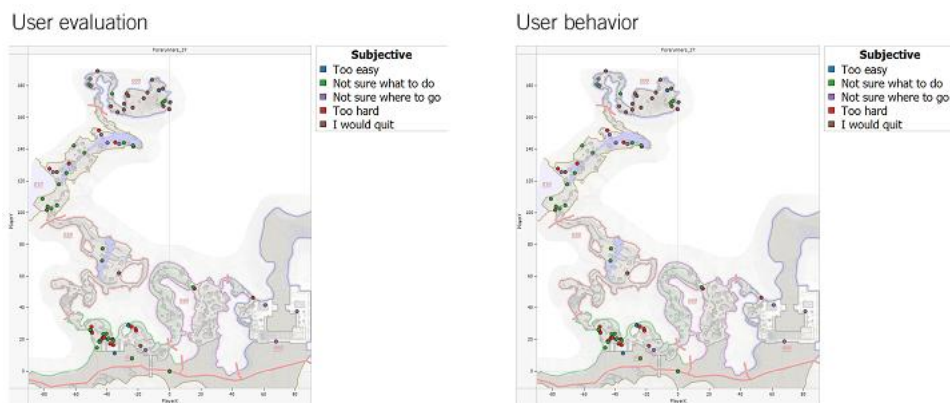


Figure 9.5: Graphical visualizations of data logs help to reveal patterns in user behavior or reactions.

Camera/Video/Audio Recording

Textual data is limited when used to inform the design of a user experience. Words rarely articulate the significance of one’s message without other supporting media types. Hence, a picture is worth a thousand words. Rich forms of data capture are necessary components of most interactive research endeavors. Their goal is to observe behaviors and emotions of potential users. As a result, researchers must be able to see what they do, where they do it, and listen to their voice as they talk about it. Without still images, video, and audio, many of these insights are not as compelling. Fortunately, many of the tools for capturing these kinds of data are easily attainable and affordable.

Post-Test Questionnaires

To capture the feedback from participants following product evaluation, a post-test questionnaire helps quantify success or failure. However, care must be taken when designing the questionnaire, because participants are more likely to report results with a positive bias. By keeping the language of the questions balanced, this bias can be mitigated, and participants’ answers are more useful and honest as a result.

Collaging

Collaging is a creative technique that empowers the participant to create a visual and literary composition that expresses their emotionally driven needs in a tangible form. It is the responsibility of the research team to create a toolkit that guides the creative process. In many cases, participants are reluctant to draw or sketch their own compositions like practitioners in the industry do. A visual toolkit can mitigate this obstacle. It can have varying degrees of fidelity and intent. This can range from magazine clippings and photos of real-world artifacts to a searchable database of digital images or online collaborative document. This may result in higher level expressions of why a product might be successful.

Scrapbooking

Similar to collaging, scrapbooking does not embody the product, but instead embodies the participant's expression about the product. It is a creative process that communicates who the participant is and how a product of interest might integrate into his or her lifestyle. Artifacts and photos from the participant's home or work sometimes come into play, and shed light into his or her daily experiences. Scrapbooking can also help stakeholders in the process empathize with the user.

User-Created Scenarios

Scenarios can be created by participants using supporting materials such as words, photos, drawings, etc. to illustrate the way to reach a desired goal or complete a task in their most familiar way. The goal is to gain a better understanding of the motivations of participants to complete a task or perform a behavior.

Data Visualization

Data visualization can represent qualitative and quantitative results that are easy to interpret. For example, trend analysis can be represented as a bar chart to provide a different perspective of the data. The practice of visualizing the information helps remove ambiguities in the data.

Prototype Capable of Adjusting Quickly

Prototypes are key to the iterative design process for Surface experiences. Design and research disciplines use prototyping to collaborate closely. The outcome is fast iteration of solutions based on immediate feedback that identifies and validates opportunities and problems.

Regression Analysis

This is a data modeling technique used to analyze numerical data. It can be used to help set parameters in Surface experiences.

Standard Set of Questions and Process

This is a technique used to design a benchmark study that can be repeated for many designs.

Scorecards

Scorecards are used to track the score of an application or component. They can then be used to determine the areas of the experience that need to be improved. Following is an example of a scorecard for two tasks performed from Launcher: launching an application and switching between applications.

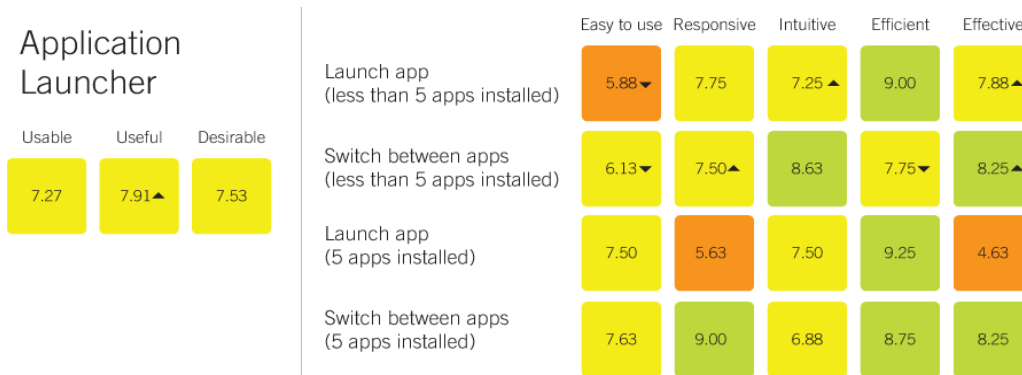


Figure 9.6: This scorecard represents two tasks performed from Launcher: launching an application and switching between applications, each under two different circumstances.

9.2 Design

Microsoft Surface experiences challenge designers to think of new ways to solve interaction and experience problems. When designing Surface experiences, it is imperative to begin prototyping, testing, and iterating in the earliest phases of the design process so that play and exploration are fundamental to the process. Adjusting existing methods and employing new ones will help lead to solutions that adhere to the philosophy and principles of Surface computing, appropriately optimized for gesture, touch, and direct manipulation.

This section outlines a toolkit of methods to help designers explore such solutions, and recommends ways to adjust the design phase to accommodate iteration.

9.2.1 Design Methods

Design methods for Surface experiences expand on existing GUI design methods and introduce additional methods for designing successful solutions that adhere to the Surface philosophy. They are flexible and are intended to be used where applicable to the design problem. The methods focus on ways to prototype quickly and cost-effectively, and to iterate throughout the entire design phase.

	Philosophy	Consideration	Method(s)
1	Spatial	Progressive disclosure	3-D simulations Hacking
2	Seamless	Physical objects	Blank device prototyping Cobbling things together
3	Social	Interactions above the table	Observing/documenting behaviors of people together Participatory prototyping Miming
4	Superrealism	Real-world behaviors	Analogous experience analysis Behavioral simulation Miming
5	Contextual Environments	Subtle affordances	Rich visualization prototyping Hacking Miming
6	Scaffolding	Narrowing down choices	Improv Experience charts Participatory prototyping
7	Performance Aesthetics	Mechanical behaviors	Behavioral simulations Hacking
8	Direct Manipulation	Fingers and things	Blank device prototyping Improv Analogous experience analysis

Figure 9.7: Following is an example of a scorecard for two tasks performed from Launcher: launching an application and switching between applications.

Cobbling Things Together

This method uses existing objects, tools, and “things” that are assembled quickly to represent a structure, behavior, or interaction. The result is an inexpensive way to perform a gesture or interaction. For example, in his book *Sketching User Experiences* (2007, Morgan Kaufmann), Bill Buxton uses an example of simulating the interaction of scrolling a map by placing a mouse on the side of a laptop and simulating the scrolling behavior. One benefit of this method is to test a potential experience or mechanism quickly and efficiently before putting resources into building a prototype.

Leveraging Quick Software

This method uses any software that is available to simulate an experience that is not built on the intended technology, in this case, Surface. One way of applying this method is to sketch with a feature in Microsoft Silverlight™ called “Deep-zoom” to explore progressive disclosure through navigation. One example is a project done for Hard Rock Café called “[Hard Rock Memorabilia](#).”

3-D Simulations

Surface experiences are spatial and immersive, and the size of the environment should always feel like it exceeds what is visibly displayed on the screen. Three dimensional simulations help designers understand the feeling of the space and the experience of moving through it. They can be created quickly in a 3-D program such as SketchUP, or by using a video camera to film the experience of walking through physical spaces.

Visual Storytelling

Fundamental to any type of effective communication, visual storytelling helps to describe an experience and takes a user through the concept and its potential. Visual storytelling can be done by sketching scenes similar to those in a comic book or a storyboard.



Figure 9.8: This set of storyboards helps tell a multi-faceted story of human interaction, device usage, and data transmission.

Other ways of visual storytelling include filming a scene that is acted out, creating a collage (placing different materials in a composition,) and recording an interaction simulated on a Surface unit.

Miming

The method of miming rehearses an interaction that is then performed with an animation so that it appears as if you are interacting with it. Bill Buxton describes it as a simple sketch animation of the interface in action. The designer then memorizes the interaction and performs it in time with the animation. When the timing is right, it appears as if he or she is interacting with the interface. Miming is an inexpensive way of exploring and also communicating a potential interaction without having to put resources into building working code.

Blank Device Prototyping

Prototyping for Surface experiences is not just about the interface. It also encompasses the physical experience a user has through touch, gesture, and direct manipulation. In order to understand the possibilities and outcomes, an early prototype is necessary to evaluate the interaction model and the engagement of the overall experience. Blank device prototyping eliminates any preconceived notions of old metaphors used in GUIs, and creates an open canvas to explore new and natural interaction solutions that solve old problems in new ways. It offers designers a cost-effective way to:

1. **Design by play:** employs the design team to explore new ways of solving interaction problems that are optimized for touch and gesture.
2. **Evaluate:** offers quick evaluation of ideas by the design team and potential users through the use of “activity spaces,” simple models, and functional interfaces.
3. **Communicate:** conveys the experience via visual solutions and tangible working models.

When designing GUIs, a paper prototype allows a designer to think through interaction problems quickly and at a low cost.

When designing NUIs, prototyping is no longer the way to test ideas on the screen. It becomes a way to test the natural behavior and gestures that direct manipulation requires. That is, it tests the “experience” in a physical way. In order to allow the physical nature of the experience to be explored, the paper prototyping method used in GUI design is extrapolated into the physical world by adding three-dimensional objects, called “blank devices.” These devices are basic three-dimensional shapes that represent both interface objects and physical objects that have been stripped of content and specific

form.

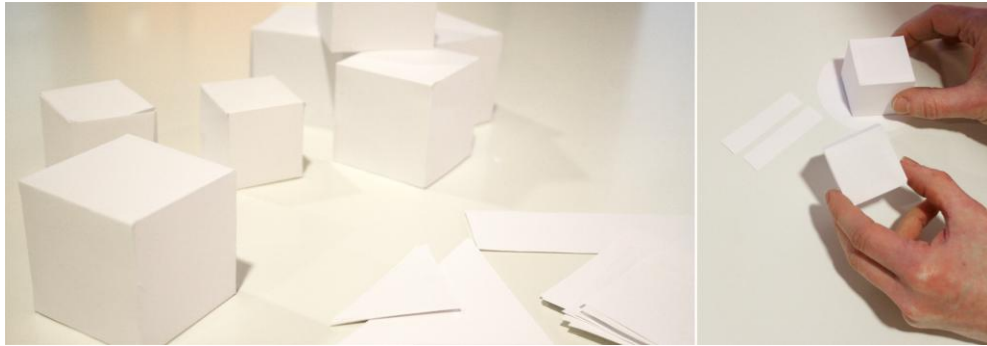


Figure 9.9: These “blank devices” facilitate spatial brainstorming and play-based design that best mimics NUI principles.

They become “bits” that can be drawn on, moved around, folded, bent, and gestured with. Removing any preconceived notions about the objects’ appearances directs the focus toward the experience and finding the most natural solutions to the interaction problem at hand. This method allows designers to begin designing through play at an early stage in the design phase. They are able to efficiently think through interaction problems, new types of gestures, and physical movements before dedicating resources to building functional prototypes.

Observing/Documenting Behaviors of Users Together

Microsoft Surface experiences are not only the interactions between users and interfaces, but also about shared experiences and social interactions between users themselves. In order to design experiences that facilitate these social interactions, it is important to observe and document behaviors of people interacting in the real world along with simulated interactions and role-playing. This approach leads to discovering serendipitous moments and inspirational social interactions for the intended design objective.

One approach to this method is to have a group of users act out an established set of situations. The nature of observing and filming their social interactions can tease out inspirational interactions and lead to an unexpected design solution.

Another method is to observe and document a group of users interacting with an interface, and focus on the conversations and interaction that the interface inspires between the users. Phrases, exclamations, sharing, and discussions can lead to a design that targets social interactions.

Wizard of Oz

This method is a way to improve or act out interactions. An example is to have a person interact with a glass table by means of dry-erase markers and blank devices. Another person “acts” as the application and provides appropriate props and responses to gestural input or user behavior. The experience is documented by video, which is used to evaluate the experience later.

Analogous Experience Analysis

Part of exploring new solutions is finding inspiration in situations and experiences that are similar to the intention or goal. The insights discovered by analyzing analogous experiences can lead to an understanding of both the visual and behavioral attributes of an experience, and can inspire different uses and situations that solve design problems in new and unique ways.



Figure 9.10: One might look to culinary activities to inspire gestures such as grating cheese as a way to manipulate content. It may inspire even more analogous experiences, in this case, using a washboard. Gestural inspiration can come from anywhere.

Participatory Prototyping

In the participatory design method, the designer becomes a researcher and is coupled with a user. The method results in many iterations on the fly based on immediate user feedback. In order to use this method effectively, it is essential for the designer to prototype and adjust the prototype quickly.

Another approach to this method is to build a prototype that can be adjusted on the fly by the participant. This way, the prototype can be sent to the participant, and the configuration can be adjusted in his or her own time. In this case, the participant temporarily becomes the designer by providing direct feedback and suggesting innovative solutions.

Behavioral Simulations

With Surface experiences, the behavior of elements in the interface includes a sense of aesthetics. The subtleties of the physics, motion, material attributes, liquid, and overall behavior are crucial to the quality of the experience and fundamental to all Surface experiences. This method uses behavior simulation software such as Havok™ to explore the behaviors and adjust the subtleties. For example, if the design solution requires a gesture that allows the user to swipe the surface with his or her hand to create “wind,” then physics and motion can be applied to objects so that they respond accordingly.

Experience Chart

Experience charts are used to map the emotions that people have throughout an experience: satisfied, surprised, dismayed, and so forth. They lead to an understanding of the emotional feedback at any point in the experience, such as anxiety, fear, happiness, fulfillment, curiosity, entertainment, or boredom. Following are two examples of experience charts:



Figure 9.11: The above image represents a day in the life of a highly mobile computer user, with various key moments (in this case interactions with technology) shown positioned around a 12 hour clock. This helps visualize the key findings from observational research like Ethnography and contextual inquiry.

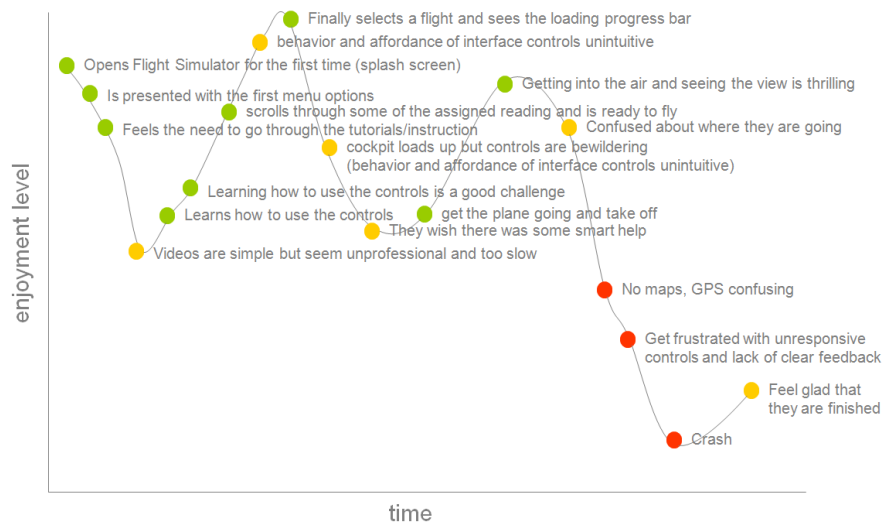


Figure 9.12: This simple chart shows peoples' enjoyment levels as they perform various activities over time in a video game.

Clearly charting out the desired experience and comparing it to the actual experience reveals opportunities for improvement. This method helps designers understand the emotional feedback so that

they can adjust the design solution appropriately.

9.2.2 Implications to the Design Flow

Designing by play requires an adjustment to the current design flow to allow more room to play and prototype, and more time to iterate frequently and at a much earlier stage. The problem with the traditional GUI software development flow is that typically each phase does not start until the previous phase ends.

The Old Way:

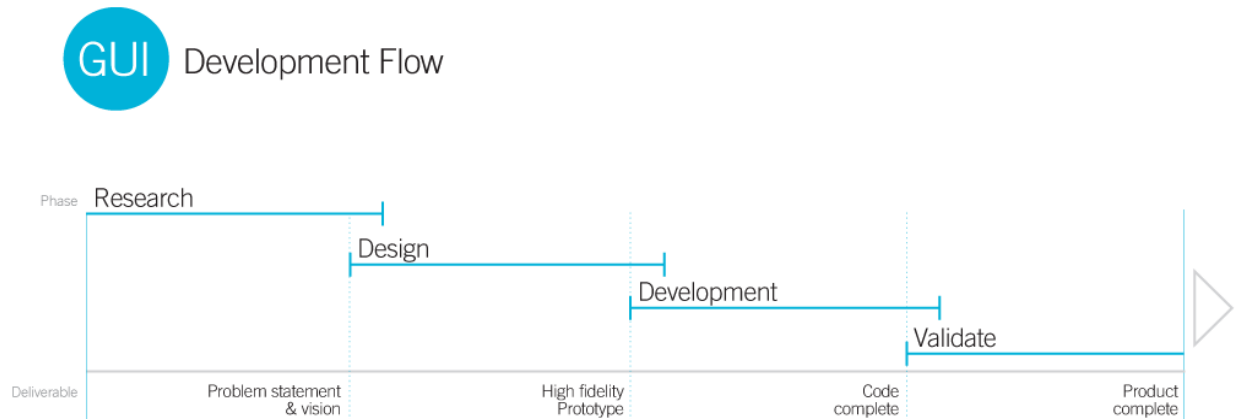


Figure 9.13: This design methodology restricts validation by testing to the last phase of the project.

While some overlap and iteration between phases exist, the traditional GUI development model is closer to a waterfall model. Only after the research phase ends does the design phase begin. The design phase ends with a high fidelity and functional prototype that is used to begin the development phase.

To design by play, it is necessary to begin the design phase earlier in the flow and stretch it out through the development phase so that prototyping can begin much earlier. When prototyping is done early, designers are able to play and explore, which leads to an understanding of the potential experience. This allows for design iterations that are tightly integrated with research, visual design, and the interaction design. It also allows the design team to play and explore as they find solutions by building many working prototypes and learning from them.

The New Way:

NUI Design Flow

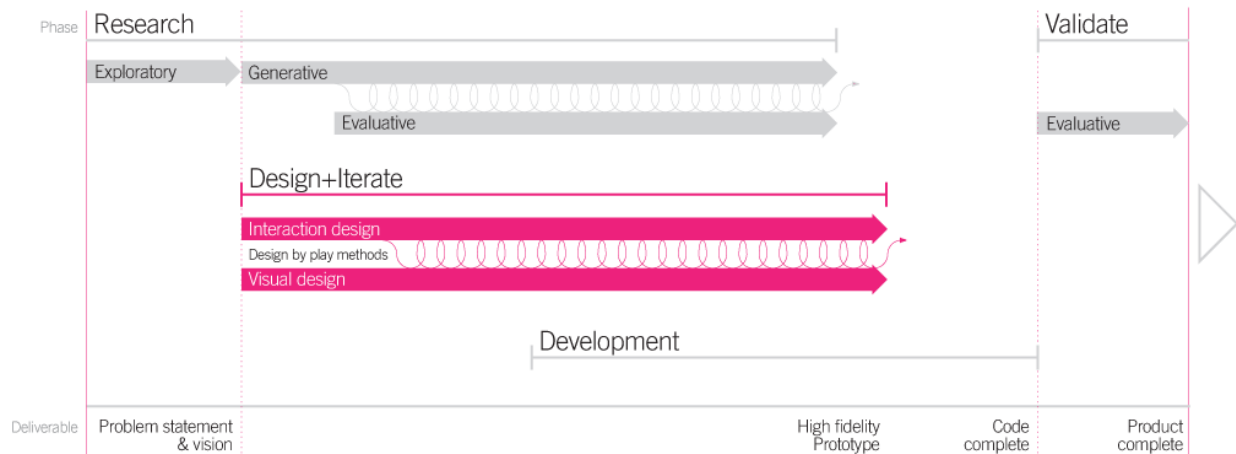


Figure 9.14: A more iterative workflow is more conducive to producing new, NUI-relevant designs through constant re-evaluation and testing.

The basic steps in the NUI design process is not fundamentally different from the way that software is designed for GUIs. The main difference is that the NUI design flow accommodates the use of design-by-play methods in the design phase so that prototyping can occur earlier and more frequently.

The entire development organization must make accommodations for the following:

- **Prototyping and iterating starts earlier in the design phase.** Since many of the interactions are based on performance and mechanics, it is imperative to start prototyping solutions quickly in order to evaluate whether an idea works.
- **Interaction design and visual design start at the same time.** Since the richness of the visuals and their performance are fundamental to the experience, both interaction design and visual design should begin at the same time. They should also be performed in collaboration to inform each other throughout the entire design phase. If a visual designer is available, then the visual design should start when interaction design starts which can help inform the overall behavior.
- **Design and research are tightly integrated.** The nature of iterative design requires close collaboration between design and research. The research phase should overlap and continue well into the design phase.
- **Development begins during the design phase.** The design and development phases blend due to the need for rapid prototyping and iterative development. This blending allows the design and development teams to discover the most successful solutions through building working models and learning from them.

10.0 References



Microsoft ThinkWeek paper: Designing Touch Experiences at Microsoft

<http://www.eamesfoundation.org>. Eames Foundation, "Powers of Ten".

Berk, Laura E. & Winsler, A. (1995). *Scaffolding Children's Learning: Vygotsky and Early Childhood Education*. National Association for the Education of Young Children

Schwartz, Barry (2005). *The Paradox of Choice*, Harper Collins

Perkins, D. N. (September 1991). What Constructivism Demands of the Learner. *Educational Technology*, 19-21.

Koster, Ralph. (2004). *Theory of Fun for Game Design*, Paraglyph.

<http://www.elotouch.com/Solutions/CaseStudies/benefitswp.asp>, Benefits of touch experiences.

Csikszentmihályi, Mihaly. (1991). *Flow: The Psychology of Optimal Experience*.

<http://www.billbuxton.com/multitouchOverview.html>, Bill Buxton, "Multi-Touch Systems that I Have Known and Loved".

Buxton, Bill. (2007). *Sketching User Experiences*, Morgan Kaufmann.

William Lidwell, Kristina Holden, Jill Butler. (2003). *Universal Principles of Design*, Rockport Publishers.

http://en.wikipedia.org/wiki/Research_methods, Research methods.

<http://memorabilia.hardrock.com>, Deep-zoom is a program available from Hard Rock Memorabilia

www.microsoft.com/usability/UEPostings/DesirabilityToolkit.doc

<http://www.usabilityfirst.com/methods/cogwalk.txt>, Usability descriptions.

(data action table) Raven, M.E., and Flanders, A. (1996). Using Contextual Inquiry to learn about your Audiences, *Journal of computer documentation*, 20(1), pp. 1-13.

<http://wilderdom.com/research/QualitativeVersusQuantitativeResearch.html>, Qualitative versus Quantitative Research.