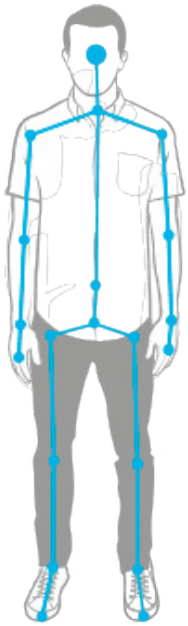


HUMAN INTERFACE GUIDELINES

Kinect for Windows v1.5.0

ACKNOWLEDGEMENT AND WAIVER.

You acknowledge that this document is provided “as-is,” and that you bear the risk of using it. (Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.) This document does not provide you with any legal rights to any intellectual property in any Microsoft product. The information in this document is also not intended to constitute legal or other professional advice or to be used as a substitute for specific advice from a licensed professional. You should not act (or refrain from acting) based on information in this document without obtaining professional advice about your particular facts and circumstances. You may copy and use this document for your internal, reference purposes. In using the Kinect software and Kinect Sensors described in this document, you assume all risk that your use of the Kinect Sensors and/or the software causes any harm or loss, including to the end users of your Kinect for Windows applications, and you agree to waive all claims against Microsoft and its affiliates related to such use (including but not limited to any claim that a Kinect Sensor or the Kinect software is defective) and to hold Microsoft and its affiliates harmless from such claims.

Kinect for Windows Human Interface Guidelines v1.5.0

Table of Contents

Introduction

Welcome to the brave new world of designing interfaces for Kinect! Successfully designing these interfaces and interactions isn't easy, but the reward is great and will delight your users and enable solutions that were either not possible or too expensive to be implemented just a few years ago. This Human Interface Guidelines document is meant to share what we've learned about designing Kinect interfaces with you and set you off on a path towards success that avoids major pitfalls we've found, highlights effective designs that we've used, and lets you focus on the unique problem that you are solving with Kinect. As we continue to learn and grow, we will update this document to reflect our findings and capabilities, with the hope that our knowledge will help make your user experience the best it can be.

Design Principles

The design principles below summarize some key takeaways from this document and are important to remember as you design your interactions.

The best user experiences are context aware

The interface should adapt as the distance between the user and the Kinect changes.

The interface should respond to the number and engagement of users.

The placement of controls should be designed based on expected user movements or actions.

Each input method is best at something and worst at something

Users will choose the input which results in the least overall effort for a given scenario.

Users tend to stick to a single input when not given a reason to change.

Inputs should be reliable, consistent, and convenient, otherwise users will look for alternative options.

Switching inputs should happen naturally, or at natural transition points in the scenario.

Confident users are happy users

It is important to keep interactions simple and easy to learn and master.

Avoid misinterpreting user intent. To increase confidence, use input combinations such as voice and gesture together.

Give constant feedback so users always know what's happening and what to expect.

The strongest designs come after user testing

Kinect enables a lot of new interactions, but also new challenges.

It is especially hard to guess what will work and what will not ahead of time.

Sometimes minor adjustments can make a huge difference.

User test often and early, and plan time into your schedule for multiple adjustments to your design.

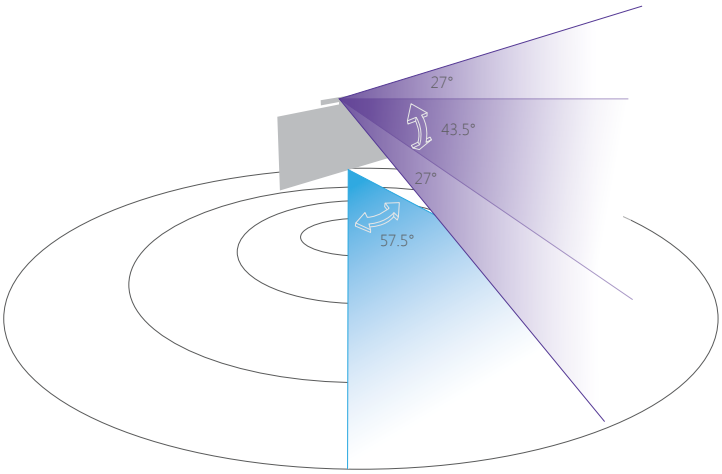
Getting Acquainted with the Kinect Sensor

What Kinect Can See

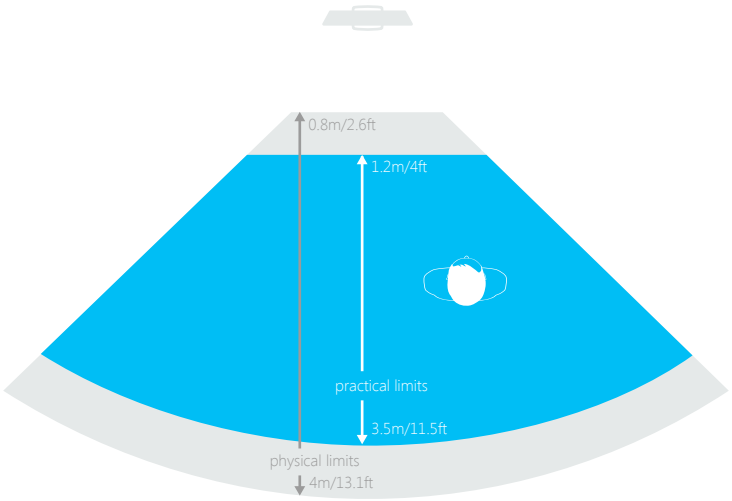
Meet the Kinect sensor: the ears and eyes of your application. Here is an overview of the sensor capabilities.

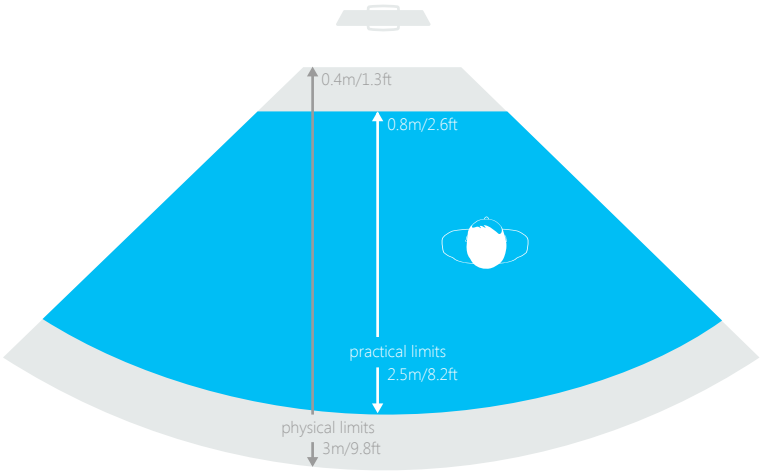
Physical Capabilities

Angles of Kinect vision (Depth and RGB)
Horizontal: 57.5 degrees
Vertical: 43.5 degrees with
-27 to +27 degree tilt range up and down



Distance ranges for Depth (default mode)
Physical limits: 0.8 to 4m
Practical limits: 1.2 to 3.5m





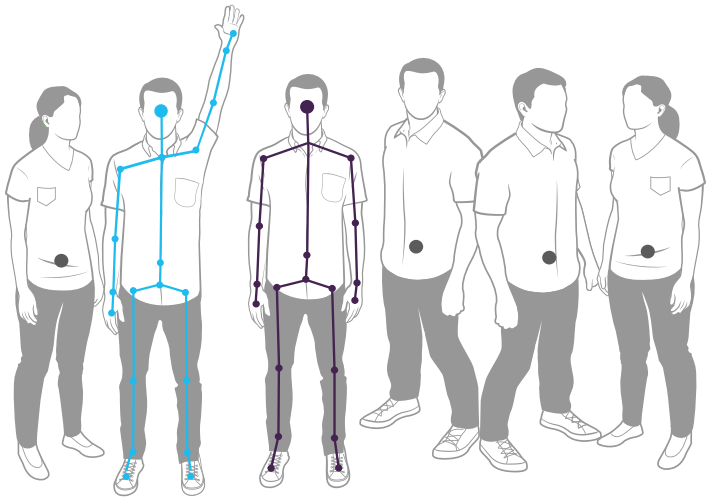
Distance ranges for Depth (near mode)
Physical limits: 0.4 to 3m
Practical limits: 0.8 to 2.5m

An application can switch between default and near mode as appropriate for the scenario.

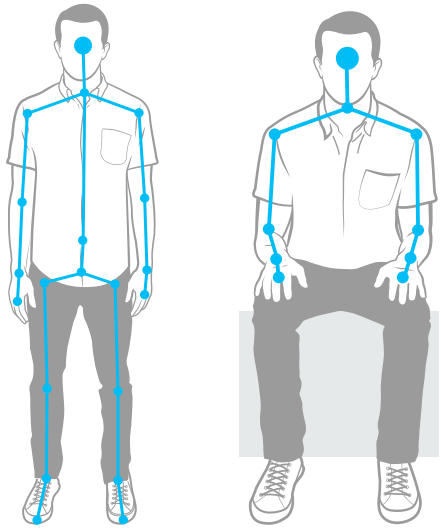
Kinect can track up to two skeletons, and detect up to 6 people within view.

Kinect can track skeletons in default standing mode with all 20 joints.

Kinect can also track seated mode skeletons with only the top 10 joints



Kinect can track two skeletons and detect up to six people.



Kinect can track skeletons in default standing mode and also track seated mode skeletons.

What Kinect Can Hear

Audio input from + and – 50 degrees in front of sensor

The array can be pointed at 10 degree increments within that range.

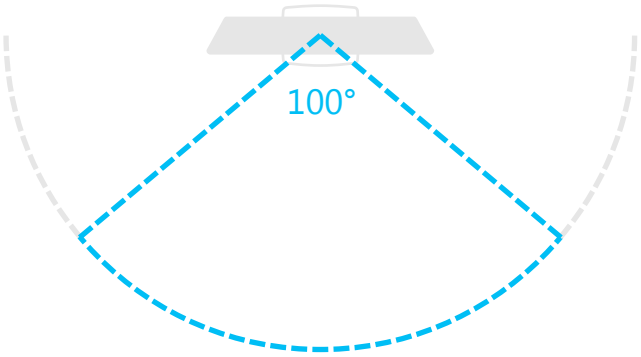
The array microphone allows 20dB of ambient noise cancelled, improving audio fidelity. That's about the sound level of a whisper.

The SDK supports mono sound cancellation, but not stereo

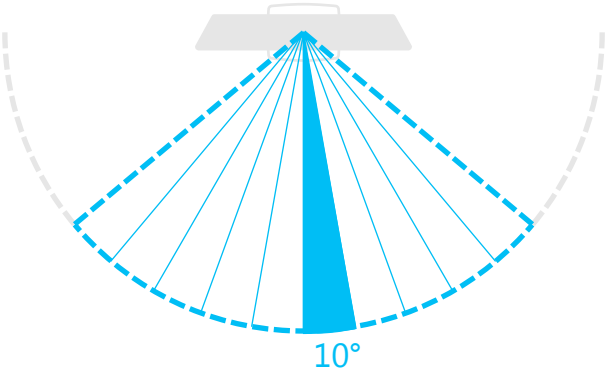
By default, Kinect tracks the loudest audio input

The microphone arrays can also be programmatically directed. For example: towards a set location, or following a skeleton as it is tracked

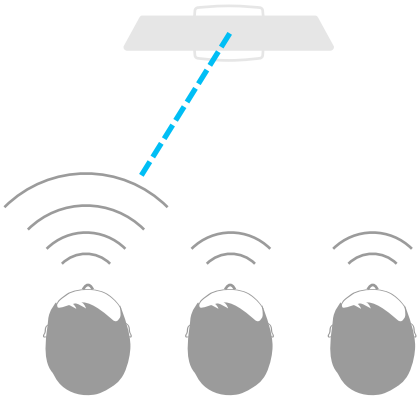
Sound coming from behind the sensor gets an additional 6dB suppression based on the design of the microphone housing.



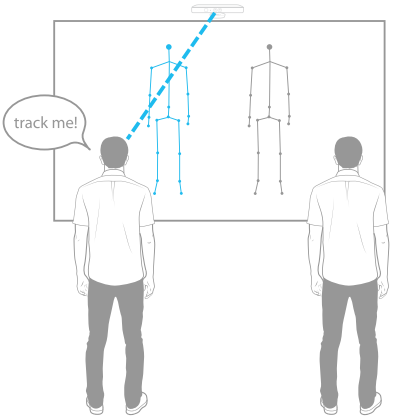
Audio input from + and – 50 degrees in front of sensor



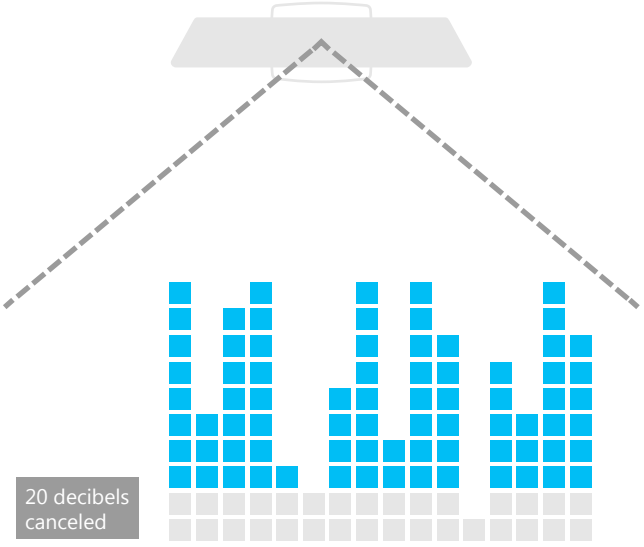
The array can be pointed at 10 degree increments within range



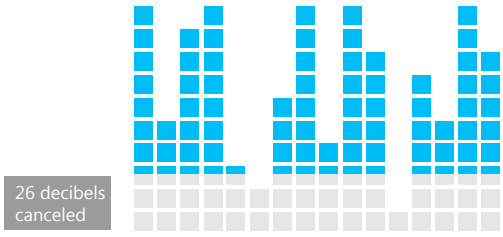
Kinect tracks the loudest audio input

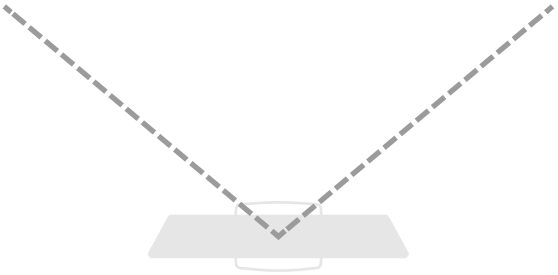


The microphone arrays can be programmatically directed. For example, following a skeleton as it is tracked



The array microphone allows 20dB of ambient noise cancelled, improving audio fidelity





Sound coming from behind the sensor gets an additional 6dB suppression

Setup and Environment Considerations

How you setup the environment where your Kinect application is used can make a huge difference in perceived reliability and usability. Controlling as many of the following factors as possible will help make your user experience shine and reduce the likelihood of disrupted interactions. Whenever possible, test your application early and often in the environment in which you plan to use it.

People and Crowds

If the environment you are designing for will have many people moving around the sensor, make sure you use the right tracking mode and design your space so that other people won't walk between the active user and the Kinect.

Ambient Noise

If you are relying on voice as an input method, keep in mind that the environment should be very quiet for it to be reliable. If you cannot control the noise level of the environment, try having the user interact closer to the device. If it is too noisy, voice may not be the right input method.

Screen Size and Resolution

Choose an appropriate display for the distance at which you expect users to interact. Remember that the size of your interface will depend on how far back your users are and what they can comfortably see.

Lighting

Avoid environments that have large amounts of natural light as it will make depth tracking less reliable. The depth camera won't work at all in direct sunlight. Dim lighting is fine for depth but will degrade the RGB image quality.

Extra Objects or Clothing

Items that drastically change the shape of a human wearing or holding them may confuse skeletal tracking. Also, items or clothing material that is reflective will interfere with the IR reflection and make skeletal tracking less reliable.

Sensor Placement

When placing the sensor, keep in mind that it is important that the sensor can see the floor for orientation. Using extreme tilt angles make tracking less reliable.

Best Practices for Designing Interactions

Gesture and Voice are two new and exciting inputs that Kinect puts at your disposal. The sections that follow describe how they work, how to best use them, challenges to expect, and how to give your users feedback that they will understand.

Gesture

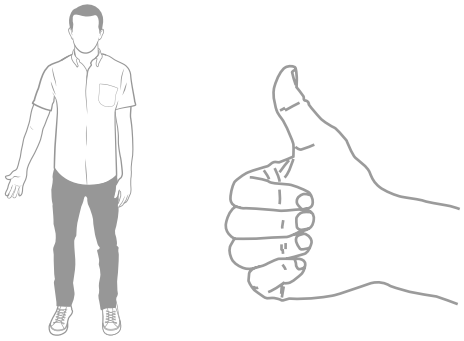
Gesture is a fun interaction method to explore, but also presents difficult challenges that need to be addressed in order to make an interface natural, usable, and appropriate for all users of your application. This section covers some basic gesture types, how to give users feedback about what the Kinect can see, and some important gesture interaction design considerations.

Basic Gesture Types

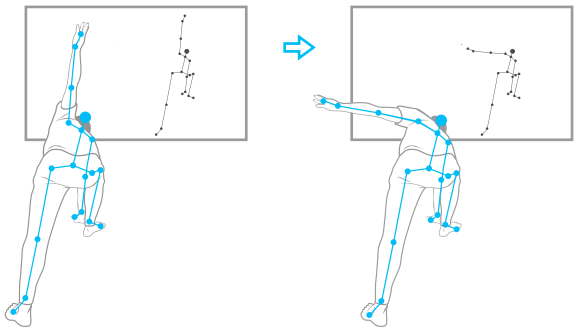
Gestures can take many forms, from using your hand to target something on the screen, to specific, learned patterns of movement, to long stretches of continuous movement. Below are some examples of commonly used gesture types and descriptions that will help you understand the gestures you are designing.

Static, Continuous, and Dynamic Gestures

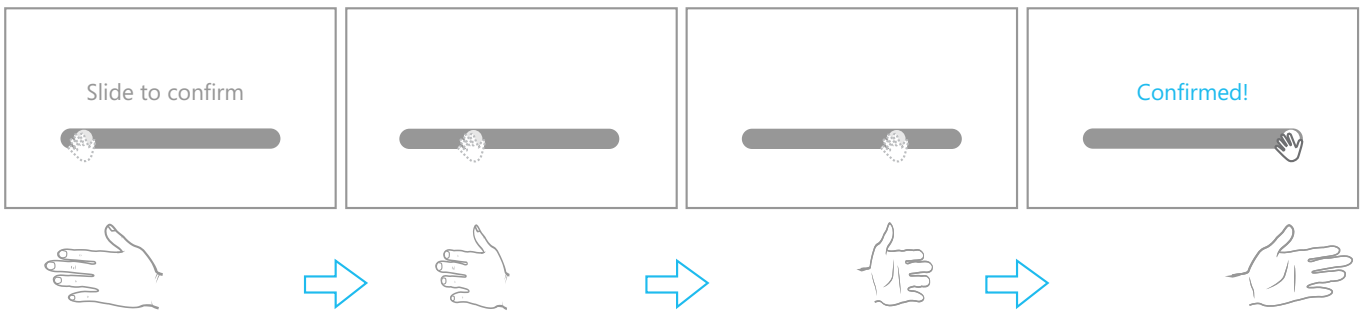
Static gestures, also called poses or postures, are ones for which the user holds a certain still position until it is recognized. Be wary of choosing symbolic static gestures such as "thumbs up" which may carry different meanings across cultures. Dynamic gestures give feedback to the user either during or after they do a defined movement. Continuous gestures track users as they move in front of the Kinect.



Static gestures are ones for which the user holds a certain still position until it is recognized



Continuous gestures track users as they move in front of the Kinect.



Dynamic gestures give feedback to the user either during or after they do a defined movement

Innate and Learned Gestures

Innate gestures are ones that the user intuitively knows or that make sense based on the users' understanding of the world.

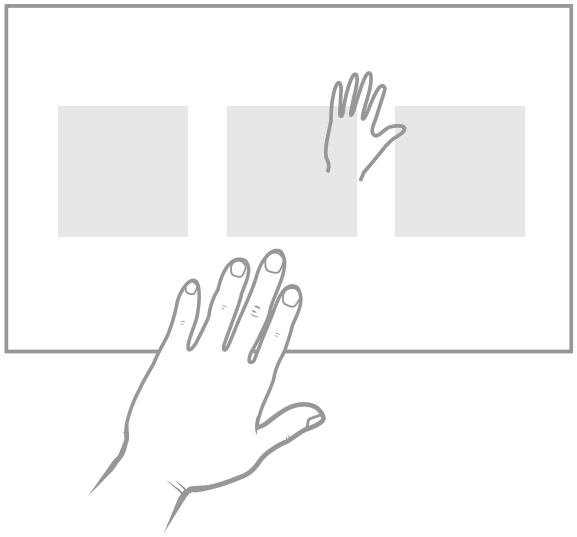
Learned gestures are ones that the user must be taught in order to know how to use them to interact with the system.

For example (innate):
Pointing to target

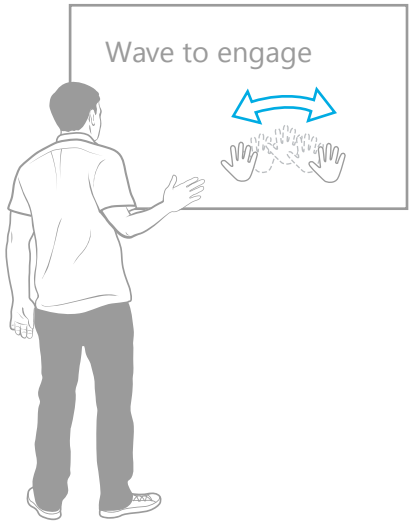
Grabbing to pick up

For example (learned):
Wave to engage

Hold left hand out to cancel



Innate gestures are ones the user intuitively knows



Learned gestures are ones that the user must be taught

Visibility Feedback

Since gesture is only effective and reliable when the user is in the correct visibility range, it's important to give them feedback to help them answer the following questions:

Is Kinect sensor on and ready?

What or how much does Kinect see?

Am I engaged and in control? Can I wave now?

Where's the Kinect's field of view?

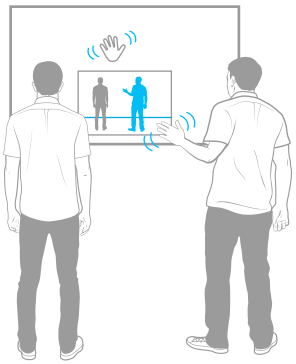
How many people can it see?

How do I know it's seeing me and not someone else?

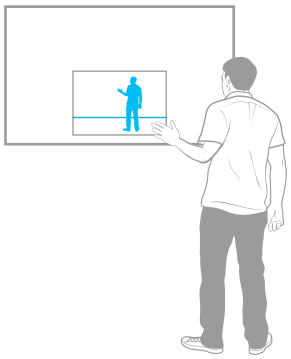
What or how much does Kinect see?

How much of the user can it see? Is the user's head in view?

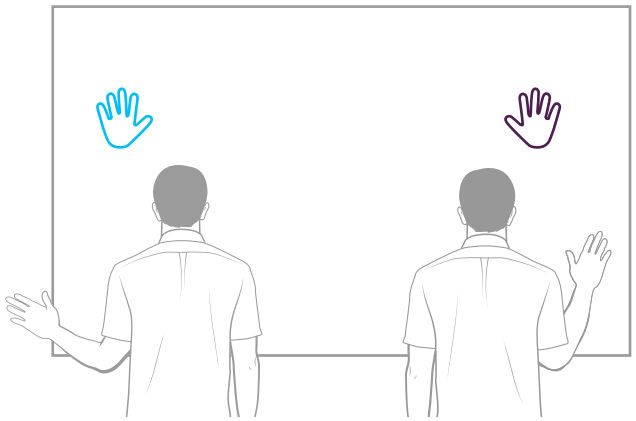
When and where can I gesture?



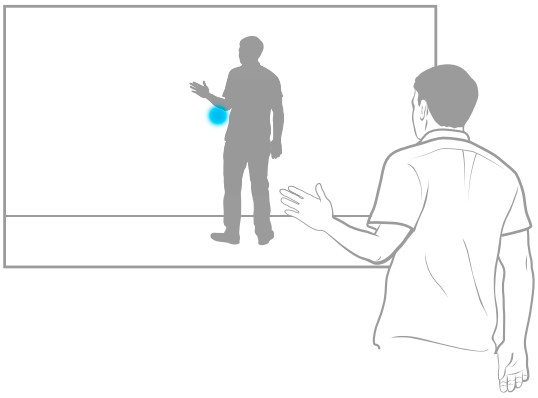
How many people can it see?
How do I know it's seeing me and not someone else?



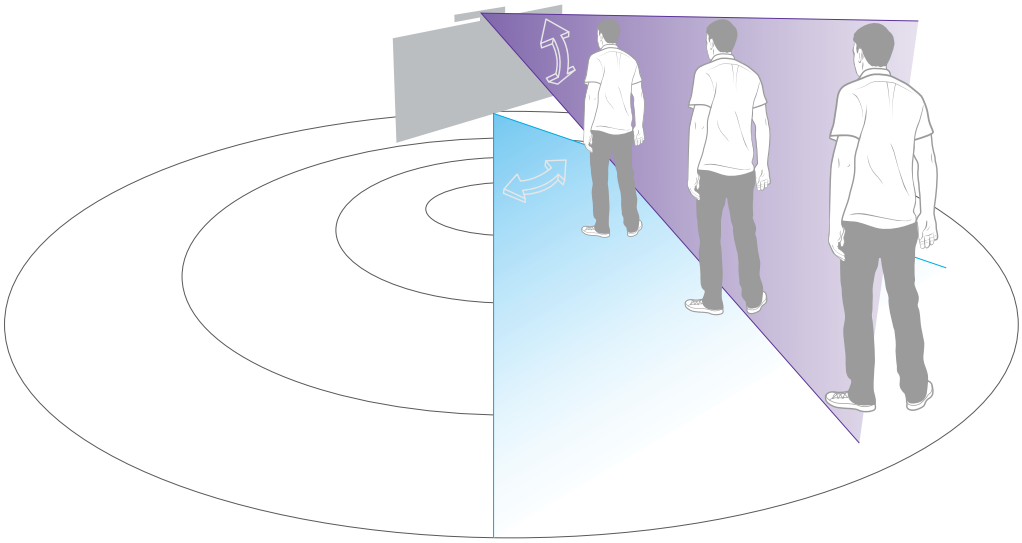
How much of the user can it see?
Is the user's head in view?



When and where can I gesture?



Am I engaged and in control? Can I wave now?



Where's the Kinect's field of view?

Gesture Interaction Design

Designing gesture interactions is a relatively new problem that the Kinect has enabled us to experiment with and begin to understand. Below are some of the key findings and considerations that we use to make our gesture designs feel magical.

Gesture Tenets

Users should agree with the following statements as they use gesture in your application:

I can quickly learn all the basic controls

Once I learn a gesture, I can quickly and accurately perform it

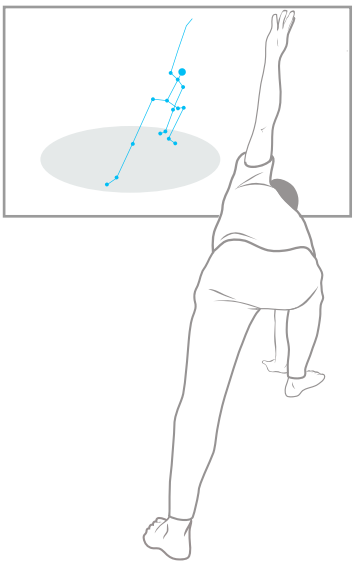
When I gesture I am ergonomically comfortable

When I gesture, the system is responsive and provides ongoing feedback

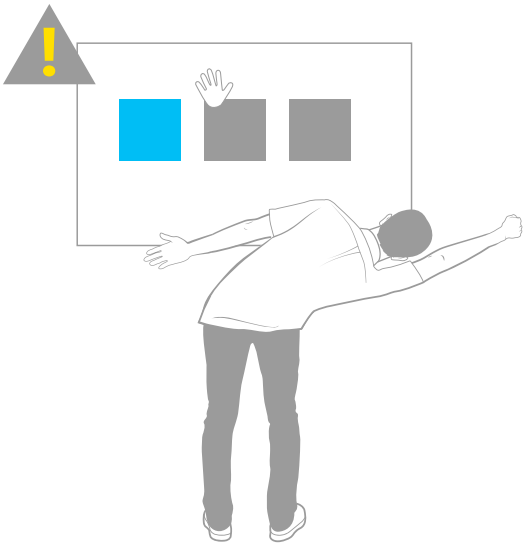
Design for Appropriate User Mindset

People still often associate Kinect with gaming, but when you are designing Kinect for Windows applications, remember: Game Mindset is NOT the same as UI Mindset

GAME MINDSET: challenge is fun!



UI MINDSET: challenge is frustrating!

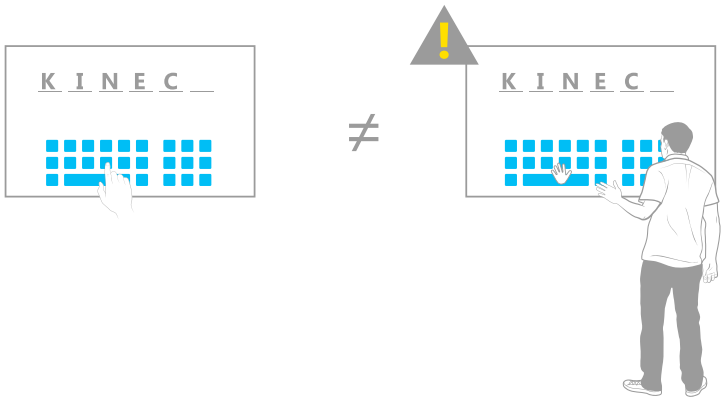


DO	DON'T
✓ If a user is in game mindset and can't perform a gesture, then it is a challenge to master it and do better next time	✗ If a user is in UI mindset and can't perform a gesture then they will be frustrated and will have low tolerance for any learning curve
✓ In game mindset a silly gesture may be fun or entertaining	✗ In UI mindset a silly gesture is awkward or not professional

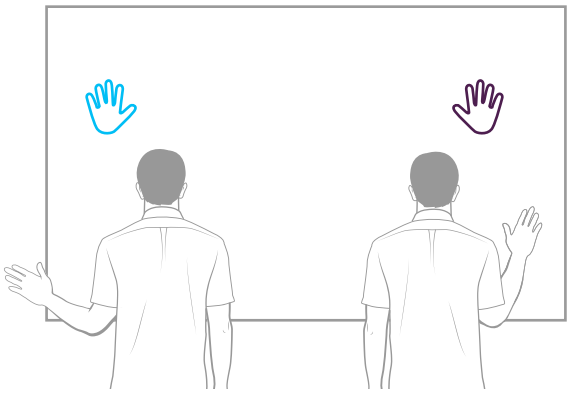
Design for Natural Interactions

Try not to force fit another form of input on an existing UI. For example, don't take a touch interface and simply map gestures to all touch input types.

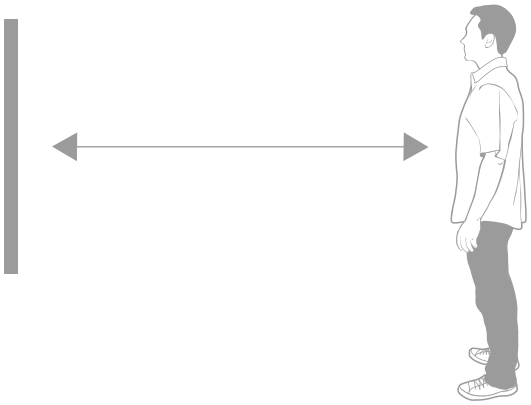
Gesture may provide a cool new method of interacting with your application, but keep in mind that its use should be purposeful.



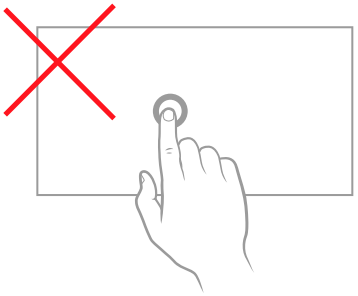
Keep in mind that gesture use should be purposeful



Multiple users will be interacting with your application at once



Scenario requires users to interact from a distance



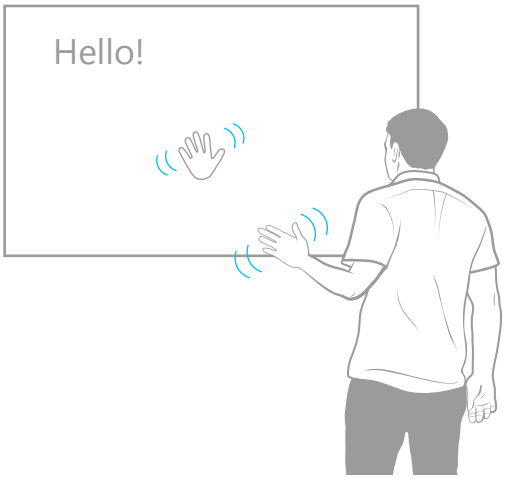
Your scenario requires that users not touch anything directly

DO	DON'T
✓ Your scenario requires users to interaction from a distance	✗ All gestures could be done faster and more easily with another input method
✓ Gestures enable an interaction that other input devices cannot	✗ Your scenario is focused on productivity, speed, and precision
✓ Part or all of the interaction is centered on unmediated, innate gestures that feel "magical"	✗ The UI is taken directly from an existing UI that was designed for a different input method

Determine User Intent and Engagement

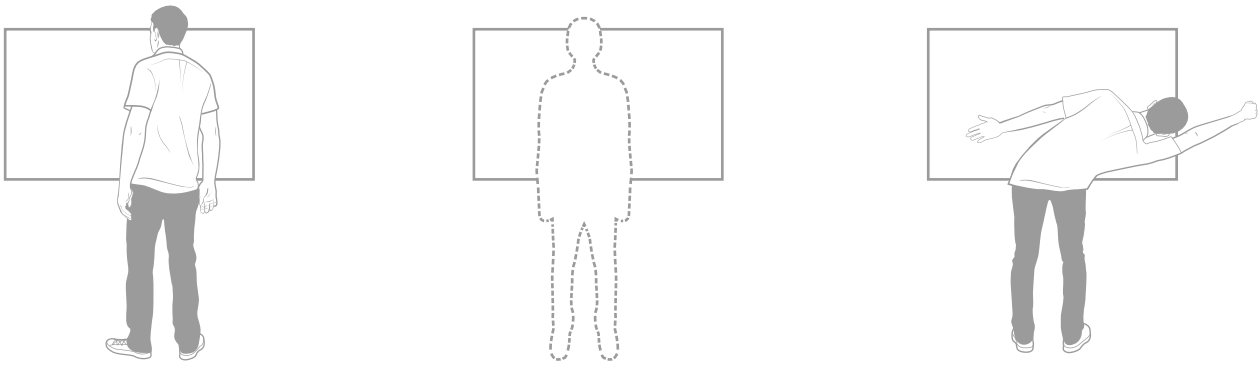
Determining user intent is the key challenge, and is a very hard problem to solve. Unlike other input devices, your user is "always on".

What if your finger could never leave the touchscreen, or your hand could not separate from the mouse?

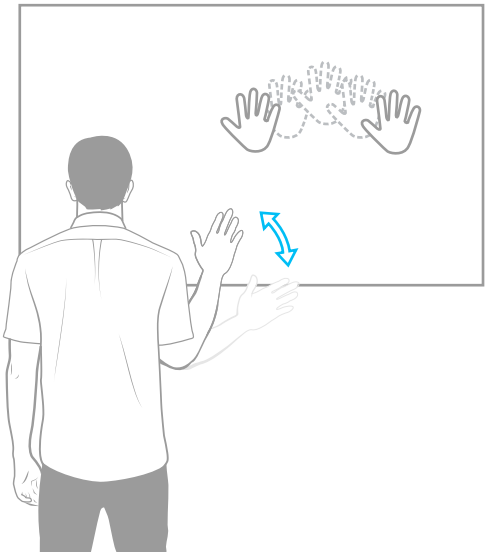


Part of determining user intent is recognizing when users wish to engage with the system. While for some scenarios, it may be desired to have the system always listening and watching, allowing users to explicitly indicate when they want to start engaging with gesture and voice will help avoid unnecessary false positive recognitions. If the user must actively trigger engagement, then the trigger must be clearly visible or demonstrated at some point in the experience.

DO	DON'T
✓ Unique and purposeful movement is recognized as gesture/interaction that other input devices cannot	✗ Natural behavior must be modified to avoid interaction
✓ Users have a clear and simple way to engage and disengage	✗ Missed Activation: I did it but system did not recognize it
✓ Natural body movements are not recognized as gestures	✗ False Activation: Natural body movements misinterpreted as the proposed gesture



Asking users to wave to engage is the method that we've used and proven as a positive way of determining user intent for engagement. Users are required to wave back and forth at least three times, rotating at the elbow for the movement to be recognized. Waving is a natural feeling yet unique movement that is not often misinterpreted. It also carries meaning to users as a way to begin an interaction.



Some other ways to recognize engagement are by distance of the user to the sensor, direction the user is facing, number of people visible, or pose. You can also choose to recognize when a user disengages to explicitly turn engagement off, rather than having a timeout or other passive method. For example: when a user faces away from the screen, take that as a cue that they are no longer engaged.

Design for Variability of Input

The experiences and expectations users bring to the system impact their interactions. Keep in mind that one user's interpretation of a gesture could be completely different from the next user.

Predicting intent is a considerable challenge since there is not a physical medium for detecting intent like a touch screen or mouse. Once the user has engaged with the system Kinect is always monitoring, looking for patterns that match a gesture.

It is critical to distinguish intentional interactions and ignore other movements
For example: touching face, adjusting glasses, drinking

To increase confidence, always provide feedback as people are using gesture.
Include an "attract" phase, an "engaged" phase, and a "completed" phase. There must also be feedback when a gesture has been canceled or failed**.**

Everyone knows what a wave is, but the way people wave varies greatly. Some people wave with their wrist, some with their elbow, and some with their whole arm. Some people wave with an open hand moving from left to right, while some people move their fingers up and down together. Simply asking a user to wave will not guarantee the same motion.

These images show two different movements that most people would recognize as waves.

Design for Reliability

Reliability should be your number one priority. Without it your application will feel unresponsive and difficult to use and your users will be frustrated.

DO	DON'T
✓ Users understand how to effectively perform a gesture	✗ Poor reliability drives users to other modes of input/interaction
✓ Users can teach or show others how to perform a gesture	
✓ Users can quickly learn how to perform a gesture and can use the gesture in similar contexts throughout the UI	

Try to strike a good **Reliability Balance**:
Reliability is affected by the design or definition of the gesture

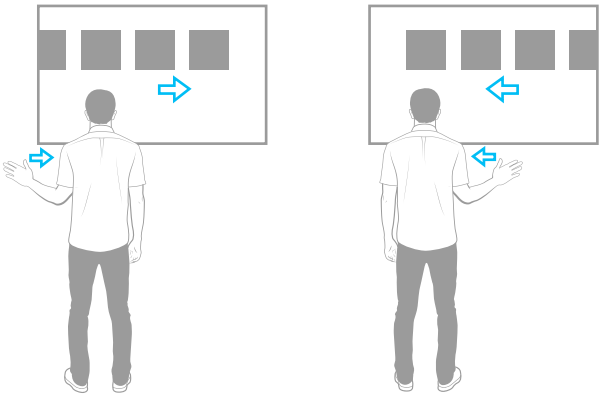
If the gesture is too rigid, there will be no false activations, but it may be hard to perform

If the gesture is too flexible, it will be easy to perform, but may have lots of false activations and/or conflicts with other gestures

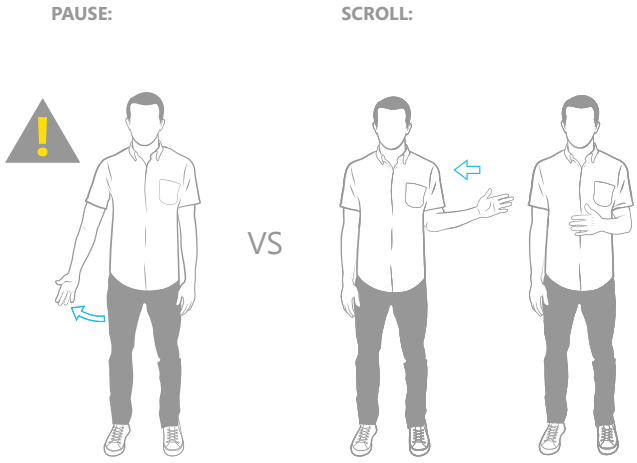
DO	DON'T
✓ The right balance considers frequency, cost of false activations, etc.	✗ If it is too rigid, users develop superstitious reasons or behaviors

Make the Gesture Fit the Users' Task

Logical gestures have meaning and relate to associated UI tasks or actions. The system's UI feedback should relate to the user's physical movement.



Move hand up to scroll content to the left



Abstract body movements that have no relationship to the task are harder to learn and remember

DO	DON'T
✓ Swipe left to scroll or move content to the left	✗ Move hand up to scroll content to the left
✓ Logical is easier to learn and remember	✗ Abstract body movements that have no relationship to the task are harder to learn and remember

Design Strong Gesture Sets

Designing a strong gesture set gets harder for every gesture you add. Therefore, it is recommended that you keep the number of gestures used in your application small, both so that they are easy for users to learn and remember, and so that they are distinct enough from one another to avoid gesture collisions. Striving for consistency with other applications will help keep things familiar for users and reduce the amount of overall gestures they have to learn and training you have to provide.

A few things to remember when defining a gesture set:
Make sure similar gestures use the same gesture language.

For example:
If you have a swipe left, use a swipe right to go the other direction.

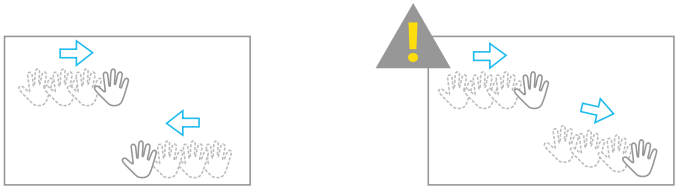
Use obvious differentiators to make gestures significantly different and reduce false recognitions and avoid overlap.

For example:
Direction changes: If the direction of two gestures is the same then it is more likely their recognition will overlap

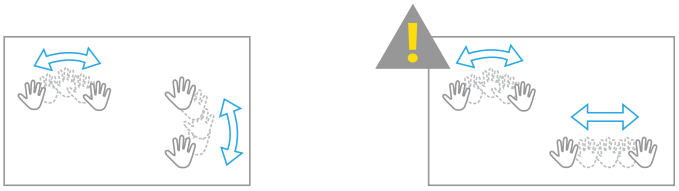
Progression: Change the path that two gestures follow, especially if it they are in the same direction

Start and end points: Start and end points are often important for recognition so try to make these vary as much as possible

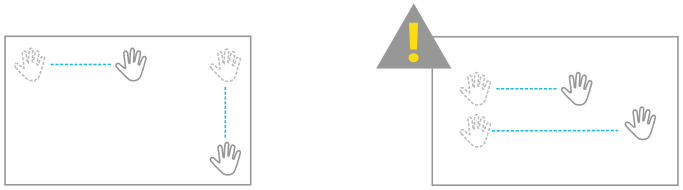
Completion time: Changing the speed with which a user must complete a gesture may be less controllable, but offers a slight difference



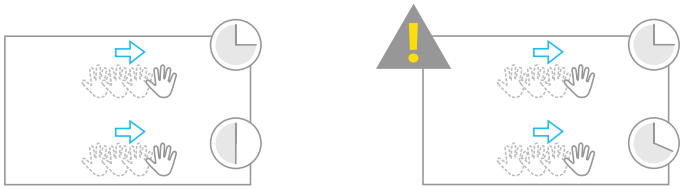
Direction Changes: Recognition overlaps if direction of gestures are the same



Progression: Change the path that gestures follow



Start and end points: These are often important for recogintion, and should be varied



Completion time: Changing the speed with which a user must complete a gesture

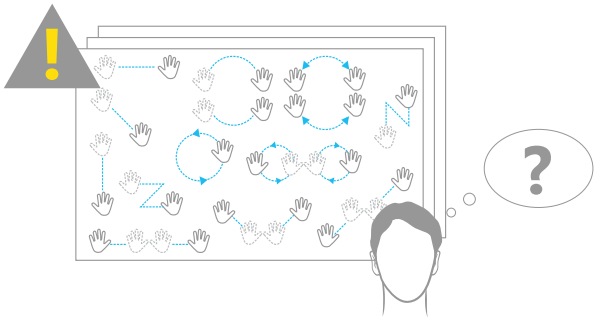
Don't overload the user with gestures to remember. Keep your user's cognitive load low. Research has shown that users can only remember up to 6 gestures.

Take cues from existing gestures that have been established in other Kinect applications

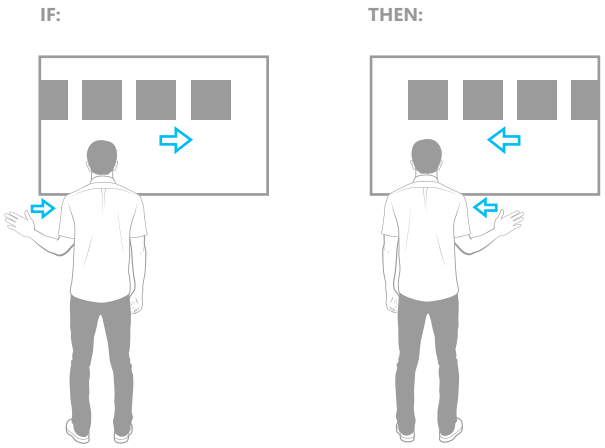
Test thoroughly for false positive triggers between gestures

Think about the complete scenario. What does the user do after completing a gesture? Might that look like the start of another unintended gesture? You can also add dead times, where all gesture input is ignored, at the end of a gesture to allow a user to reset to a neutral position before starting the next action.

Each gesture in an application's gesture set should feel related and cohesive.



Think about the complete scenario....
What does the user do after completing a gesture?



Make sure similar gestures. If you have a swipe left, use a swipe right to go the other direction

DO	DON'T
✓ Users can easily recall gestures	✗ Swipe left or right to move content left or right, but circular motions for up and down
✓ Users can generalize from one gesture to others and find them easy to learn	
✓ Swipe left to move content to the left, means swipe right, up, down to move content right, up, and down	

Handle Repeating Gestures Gracefully

Will users want or need to perform the proposed gesture repeatedly, in a back-to-back fashion?

For example:
Multiple swipes to quickly go through multiple pages/slides of content

Watch out for some common challenges:
Ignoring the "return" portion of a repeated gesture

Designing the opposite gesture, which resembles the "return" portion of the first gesture

DO	DON'T
✓ Repeating gestures are designed to be fluid, without jarring starts and stop	✗ UI feedback gets in the way and artificially hinders or paces the rhythm. **For example:** user feels like they must wait for an animation to finish before they can repeat the gesture
✓ Users can quickly get into a rhythm of movement	✗ Design of gesture makes repetition inefficient

Avoid Handed Gestures

Handed gestures are ones that can only be done with a specific hand.

For example:
Only left hand can do gesture A. The need for them usually comes from technical need to improve performance.

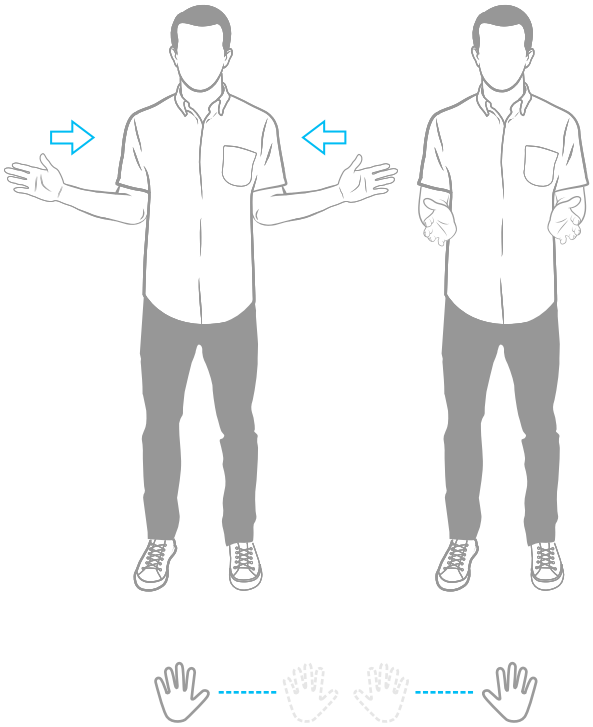
Our best advice? Make all gestures work with either hand.

DO	DON'T
✓ Users can switch hands to reduce fatigue	✗ Handed gestures: poor discoverability and accessibility
✓ Accommodate both left and right-handed people	

Choosing between one-handed and two handed gestures

One-handed gestures are preferred and make for a better efficiency and accessibility. Two-handed gestures should be symmetrical because they are easier to perform and remember.

One-handed gestures are also easier for users to discover learn and remember

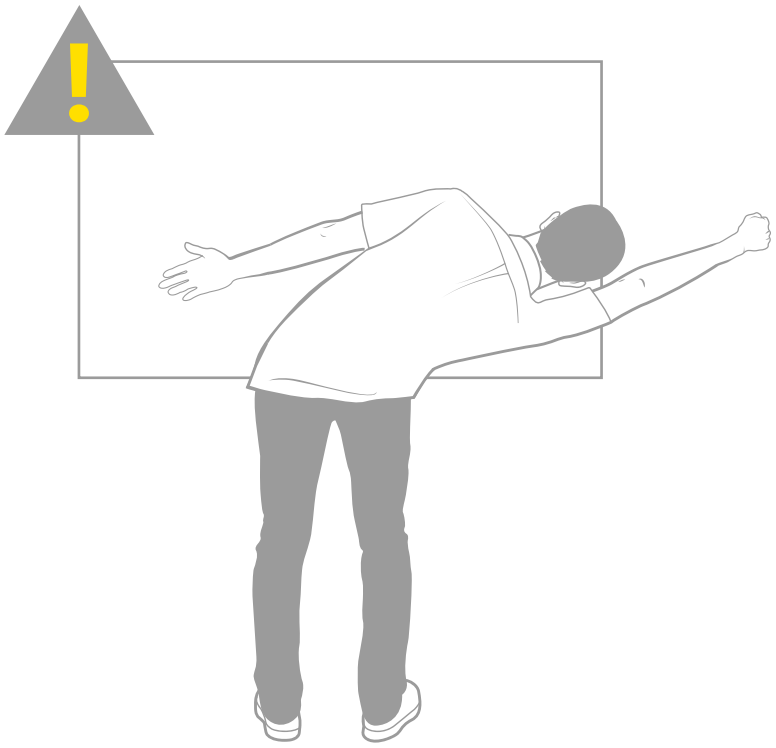


Two-handed gestures should be symmetrical because they are easier to perform and remember

DO	DON'T
✓ One-handed gestures for all critical path tasks	✗ Two-handed gestures for critical, frequent tasks
✓ Two-handed gestures for advanced-user, non-critical tasks	

Remember that Fatigue Kills Gesture

If your user gets tired because of a gesture, they will have a bad experience and will probably quit.

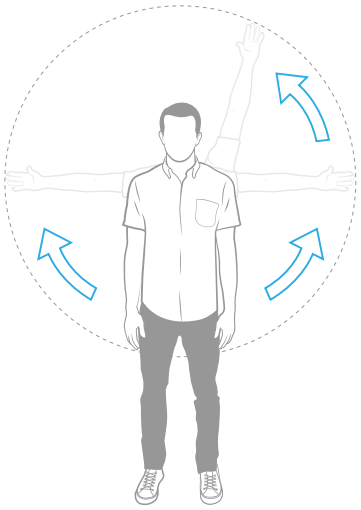


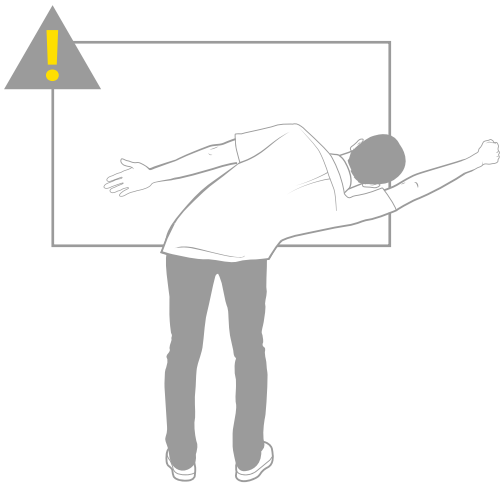
Fatigue increases messiness → poor performance → frustration → bad user experience

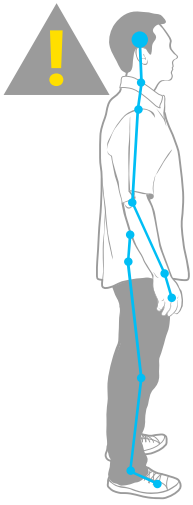
DO	DON'T
✓ Offer alternatives: list navigation (small/big steps), list jumping, zooming in/out, list reduction or filters	✗ Excessive back-to-back repetition of a single gesture, like page swiping to get through a long list
	✗ Hold arm up to scroll up through a long vertical list

Consider User Posture and Movement Ranges

User posture may affect design of a gesture, so consider where and how your users will use your application. For example, a sitting posture will limit a user's movement. Also keep in mind the normal and comfortable ranges of interaction for the human body.







Also, be aware of the loss of reliability if you are viewing a user from the side. Joints not in full view will be placed in predicted locations by the Kinect skeletal tracking and therefore are not always in the correct position.

DO	DON'T
✓ Single gesture designed to work well across all viable postures	✗ One command, 2 gestures: different gestures for seated and standing
	✗ One gesture that works well in one posture and poorly in another

Teaching Gestures and Discoverability

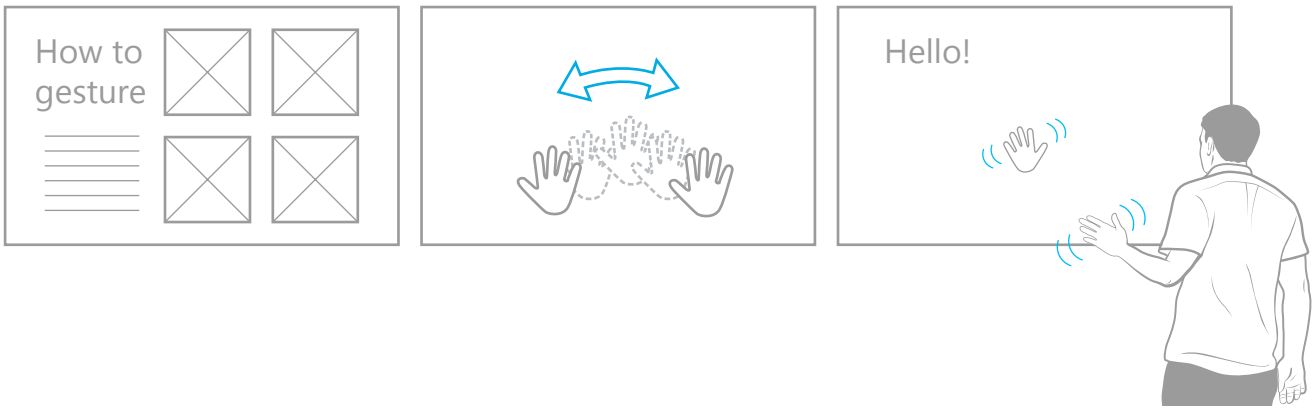
Whenever you have a gesture available as a way to interact with your system, you should find a way to communicate it to your user. There are many forms of this including:

A quick tutorial for new users

A visual cue when a user first engages, or throughout the interaction

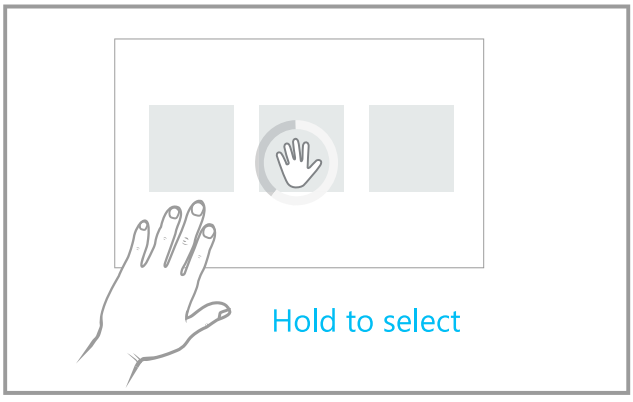
A static image

An animation

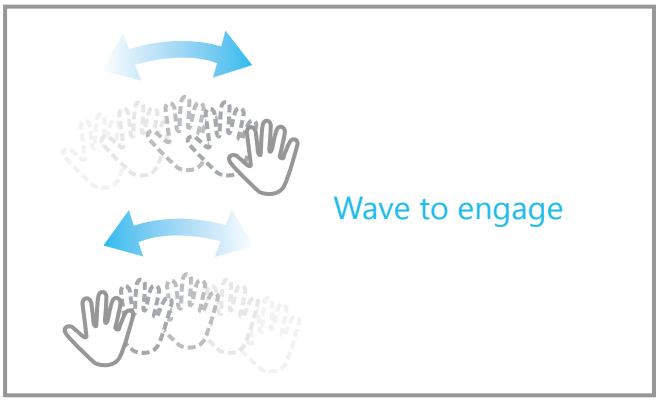


Use a method that is appropriate for the gesture you are teaching. If the gesture is static, then use a static image. If the gesture is dynamic, then use motion or animation to illustrate it for your users.

STATIC



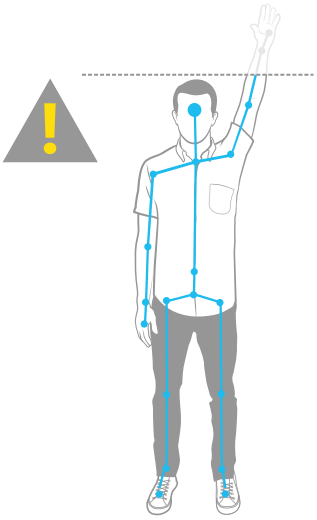
DYNAMIC



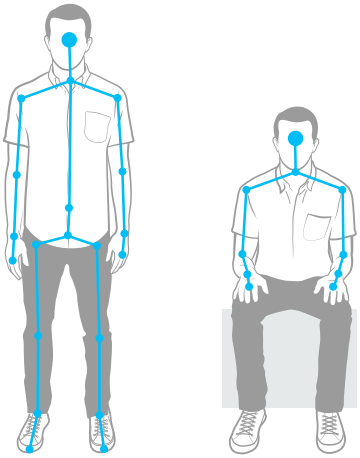
Be Aware of Technical Barriers

If you are using skeleton data to define your gestures, you should be aware of its limitations and situations where that data is less reliable.

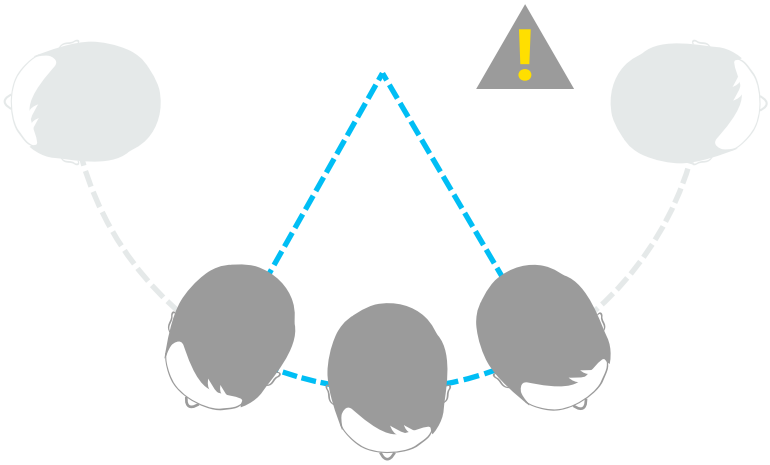
Keeping arms and hands to the side of the body when performing gestures makes them easier to track, whereas hand movements in front of the body may be very unreliable



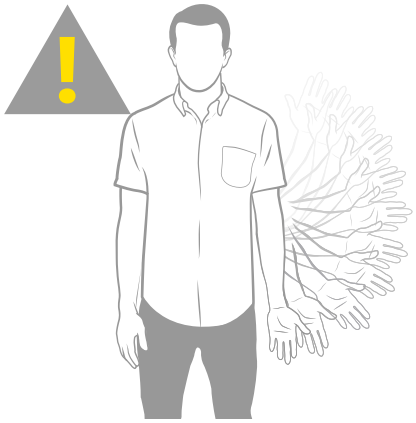
Extending hands above the head may result in the points being lost, if the Kinect field of view is not specifically set to see above the user's head



There are 20 joints that can be tracked for default mode and 10 for seated mode, neither of which include fingers.



Skeleton tracking is most stable when the user is facing the sensor



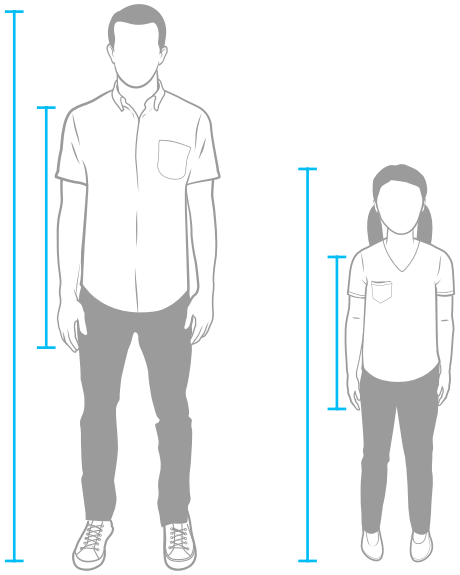
Skeleton tracking speed and frames per second limitations should be considered for very fast gestures.

Validating and Audience

Regardless of how you define your gestures, keep your target audience in mind so that gestures work for the required height ranges and cognitive abilities of your users. Think about the whole distance range that your users may be in, angles that users may pose at, and various user height ranges that you wish to support.

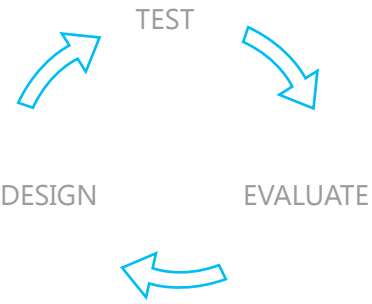
For example, if your application will be used by adults and children, you will need to account for various heights and limb lengths. Children also make very different movements than adults when performing the same action due to differences in their dexterity and control.

Children will tend to make faster, wilder, or more exaggerated movements for the same gesture.



Parameterize and Iterate

Getting a gesture to feel just right may take many tweaks and iterations. Parameterize anything you can and user test often.



DO	DON'T
✓ Gesture works reliably for your whole target user range	✗ Gesture works for you but no one else
✓ Gesture is natural and discoverable	✗ Your app is not built to be flexible so it is hard to make adjustments

Voice

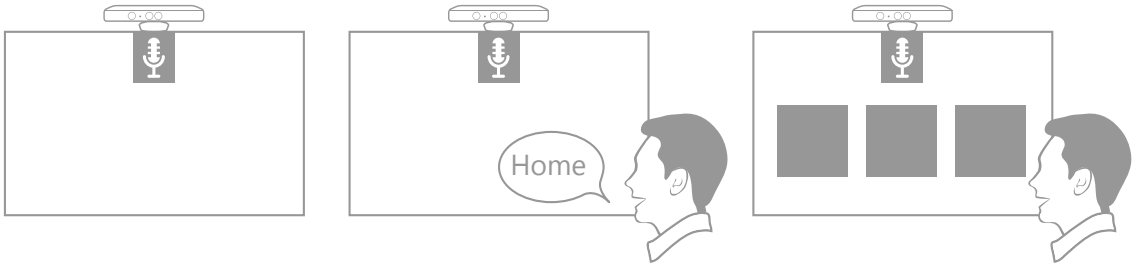
Voice is another input that can enable new and natural-feeling experiences, but it is still limited in terms of what is supported. Using voice in your application allows you to choose specific words or phrases to listen for and use as triggers. Words or phrases spoken as commands is not conversational and may not seem like a natural way to interact, but when voice input is designed and integrated well it can make experiences feel fast and increase your confidence in user intent.

Engagement Models

There are two main engagement models for using voice with Kinect:

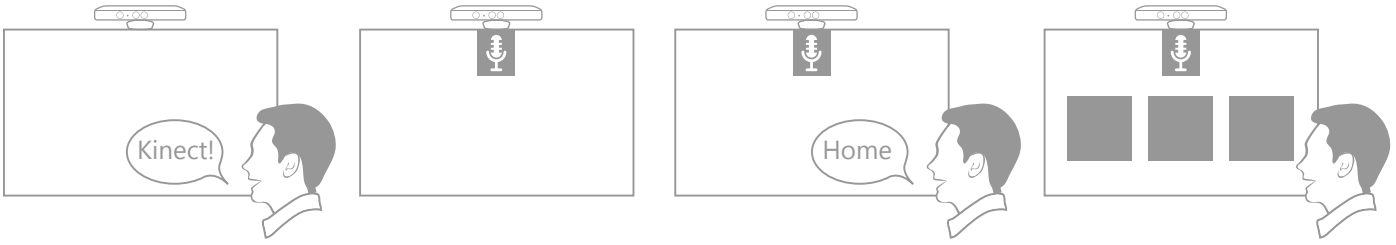
Always on, Active Listening

The Kinect is always listening for all of your defined words or phrases. This will work fine if you have a very small number of distinct words or phrases, but the more you have the more likely it is that you will have false recognitions. This also depends on how much you expect the user to be speaking while the application is running, which will most likely depend on your specific environment and scenario.



Keyword/Trigger

The Kinect is only listening for a single keyword or trigger word. Once it hears that word it will listen for additional specified words or phrases. This is the best way to reduce false positives. The keyword you choose should be very distinct so that it is not easily misinterpreted. For example, on Xbox360, "Xbox" is the keyword. Not many words sound like "Xbox" so it is also a strong keyword.



Choosing words and phrases

When choosing what words or phrases to use there are a few important things to keep in mind:

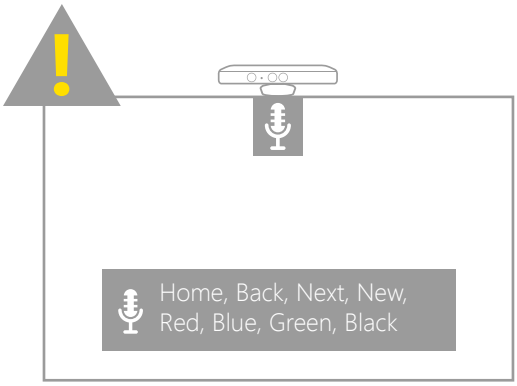
Use simple, common words where possible for a more natural feeling experience and for easier memorization

DIAPHANOUS	LIGHT
VERDANT	GREEN
ZENITH	TOP

Keep phrases short (1-5 words)

"SHOW ME THE ONE ON THE LEFT"
"PUT THIS IN THE SHOPPING CART"

Keep the number of phrases or words per screen small



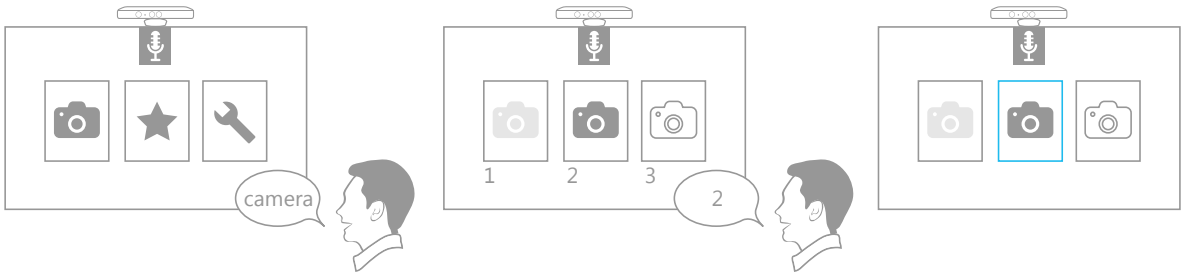
Be wary of one syllable words since they are more likely to overlap with others

CAT
DOG
FUN
TOP

Avoid alliteration, words that rhyme, common syllable lengths, common vowel sounds, same words in different phrases

Alliteration	Rhyming	Common Syllables	Common Vowels	Same words, different phrases
CAR	MAKE	FERRY	MARE	ONE MORE TIME
KITE	SHAKE	BABY	CARE	SHOW ME MORE
CORE	RAKE	WATER	FARE	MORE RED
COOL	TAKE	COOKIE	PEAR	NO MORE RED
KEY	FAKE	CARSEAT	RARE	PLAY MORE LIKE THIS

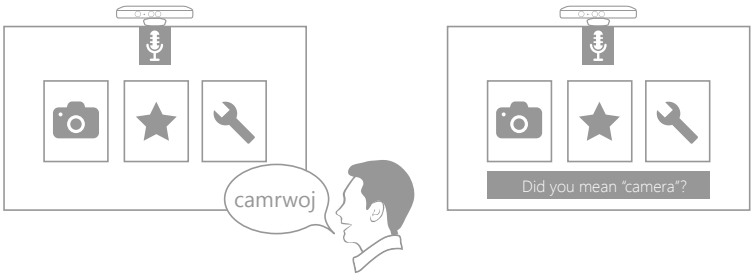
If things get too complicated, or for non-text based content, consider using numbers to map to choices on a screen.



say: "Camera, Star, or Wrench"

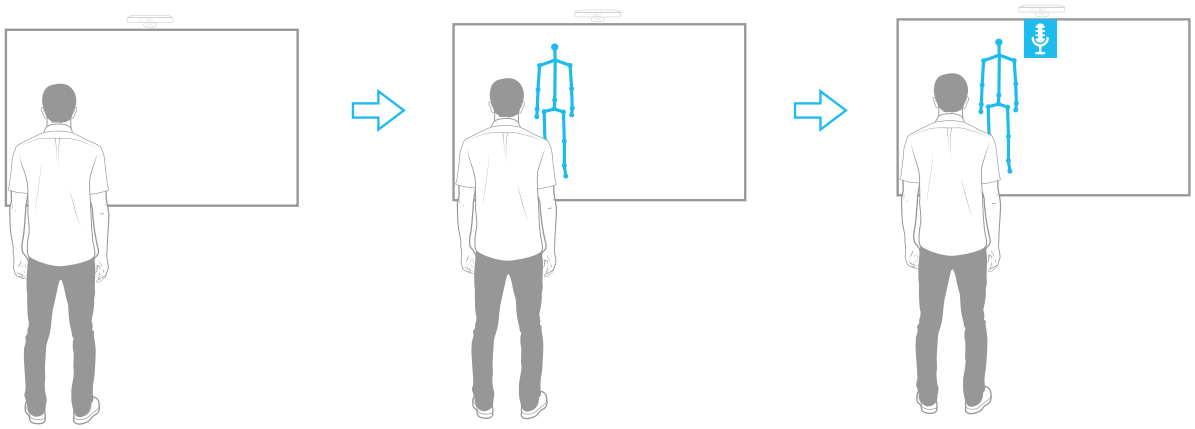
say: "Camera" then say "2"

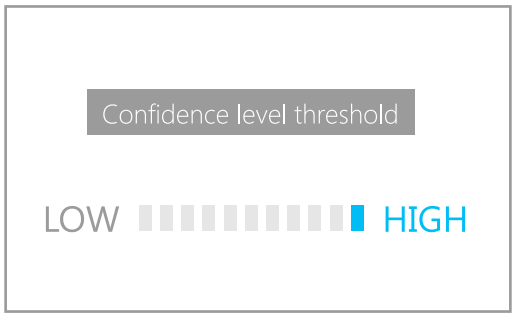
For commands recognized with low confidence, help course correct by providing prompts.
For example: "did you mean 'camera'?"



Test your words and phrases in the acoustic environment you intend the application to be used to validate that it works

Try using a trigger or event to make the system start listening
For example: only listen when a skeleton is detected in a certain area





Tune the confidence levels that you accept to reduce false activation. This may result in lower recognition rates, but could be less frustrating for users.

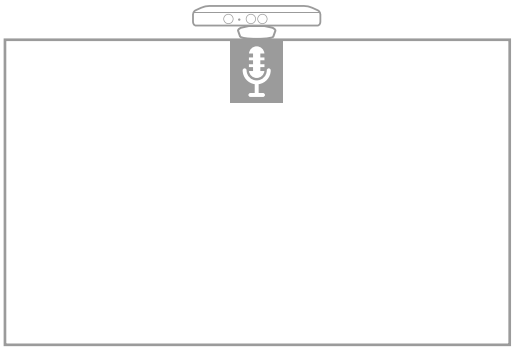
Test and be flexible. If a specific word always fails or is falsely recognized, try to think of a new way to describe it

Designing User Interfaces for Voice

Although it is possible to try and force users to memorize phrases or discover them on their own, it is generally not recommended. Here are a few user interface best practices to follow that will help users understand that they can use voice and what words or phrases are available.

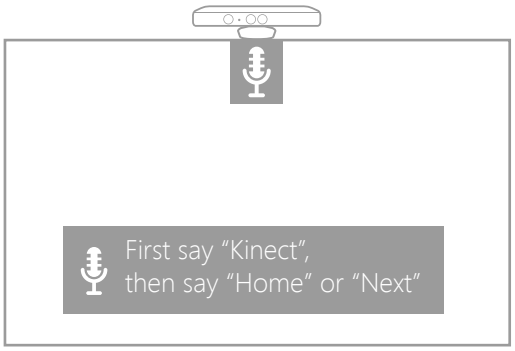
Basic Guidelines:

If there is an available voice command, the user should be able to see it written on the screen exactly how they have to say it for it to be recognized

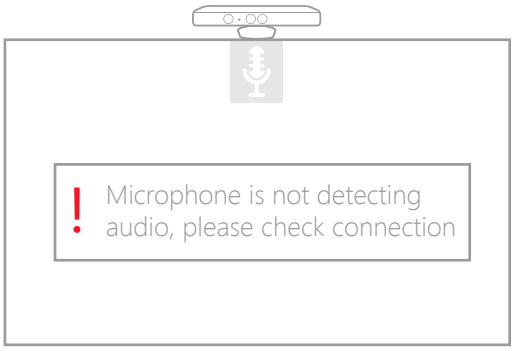


Indicate visually that the application is “listening” for commands.

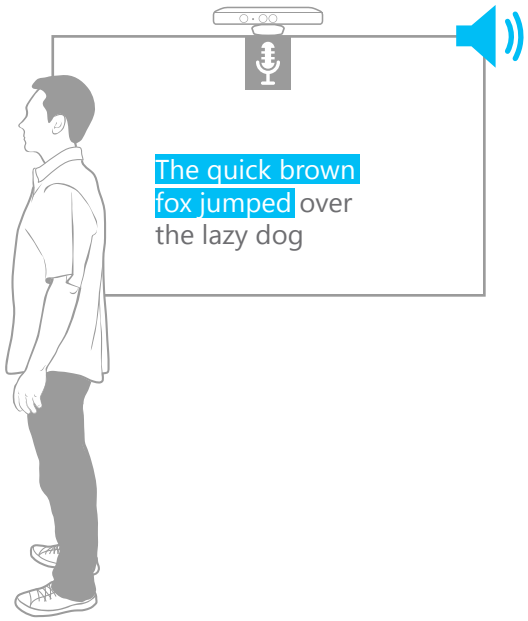
For example: Xbox360 uses a microphone icon to show this



If you use a keyword, display that keyword on the screen, or take the user through a tutorial in the beginning



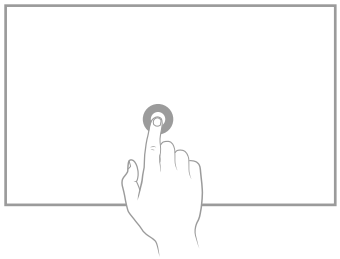
If the interface should be voice enabled but there is an issue with the microphone connection, display an error icon or message so that the user is aware and can fix the problem

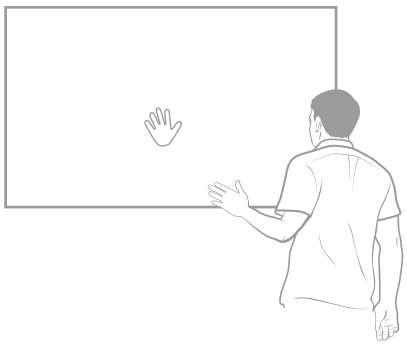


If for some reason the user may not be facing the screen, have an option to read the available phrases to them out loud

IF:

THEN:





Voice should not be the only method by which a user can interact with the application. Build in allowances for the user to use another input method in case voice is not working or becomes unreliable

Choosing the right environment for voice

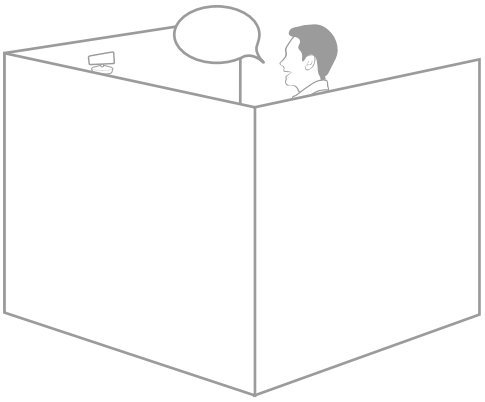
There are a few environmental considerations that will have a large effect on whether or not you can successfully use voice in your application.

Ambient Noise:

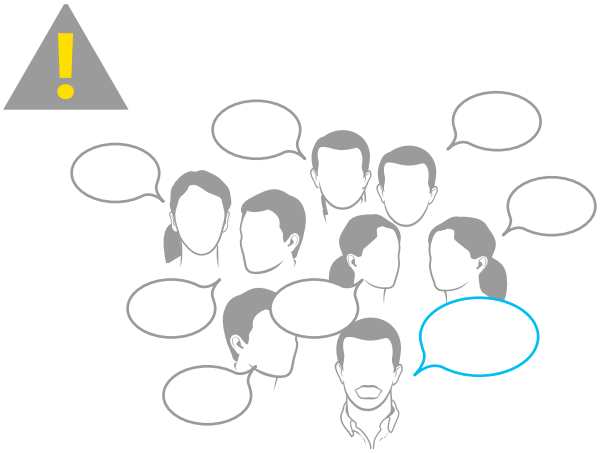
The Kinect will focus on the loudest sound source and attempt to cancel out other ambient noise. This means if there is other conversation in the room (usually around 60-65 dB) then that will reduce the accuracy of your speech recognition.

Amplify that to the sound level of a mall or cafeteria and you can imagine how much harder it is to recognize even simple commands. At some level, ambient noise is unavoidable, but if your application will be running in loud environments then voice may not be the best interaction choice. Ideally you should only use voice if:

60-65 dB =



The environment is quiet and relatively closed off, like an office.



There will not be multiple people speaking at once.

System Noises and Cancellation

Although the Kinect sensor is capable of more complex noise cancelling if a developer should wish to build that support, Kinect for Windows' built in functionality only cancels out mono sounds, like a system beep, but not stereo. This means even if you know that your app will be playing a specific song, or that song will be playing in the room, you cannot noise. On the other hand, if you are using monotone beeps to communicate something to your user that should be acceptable.

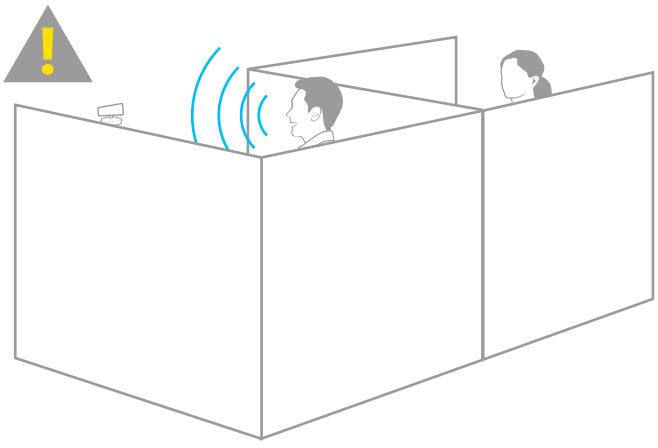
Social Considerations

Keep in mind the social implications of your users needing to say commands loudly while using your application. Make sure that the commands are appropriate for the environment, so you are not forcing your users to say things that will make them uncomfortable, and that the volume at which they have to speak is appropriate for your environment. For example, speaking loud commands in a cubicle based office setup may be distracting and inappropriate.

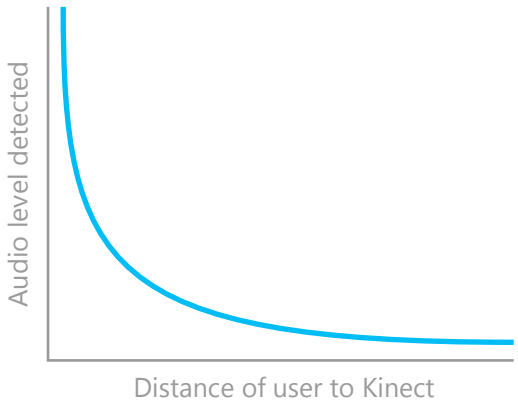
Distance of users to Kinect

As the user moves further away from the Kinect, the sound level of their voice decreases quadratically. This means that when the user is extremely close to the sensor, the sound level will be very high, but as they move away it quickly drops off and becomes hard to hear, which could result in unreliable recognition or the need for them to speak significantly louder. Ambient noise also plays a role here in making it harder for the Kinect to hear someone as they get farther away. You may have to make adjustments to find a "sweet spot" for your given environment and setup for where a voice of normal volume can be picked up reliably.

In an environment with low ambient noise and soft PC sounds, a user should be able to comfortably speak at normal to low voice levels (49-55 dB) at both near and far distances



Speaking loud commands in a cubicle based office setup may be distracting and inappropriate.



As the user moves further away from the Kinect, the sound level of their voice decreases quadratically.

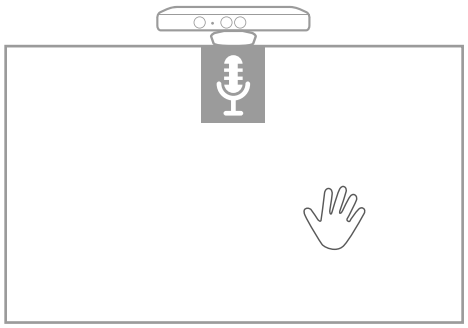
Feedback

Whether using gesture, or voice, or both, providing good feedback is critical to making users feel in control and helping them understand what's happening. Especially if they are standing at a distance and have less direct contact with the interface, it is important to take extra care in showing them how their actions map to your application. This section will cover a few ways that you can make your feedback as strong as possible.

Basic Visual Feedback

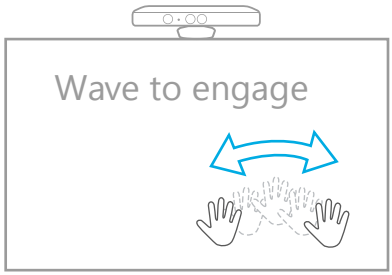
Make it clear what's actionable and how to take action

Use iconography, colors, or tutorials to show users how to differentiate between actionable controls and editorial text and content



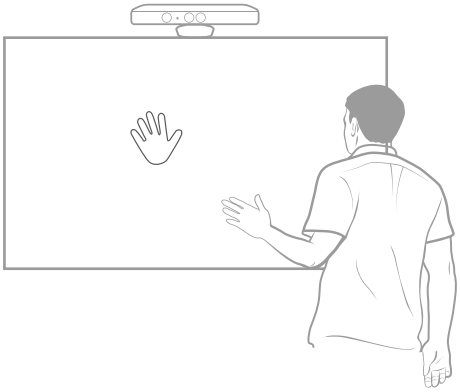
Use iconography or tutorials to show users what input methods are available to them

Show what commands are available to say using voice



Show what gestures are available.

If there is any error with Kinect state or connection, show error message or state and a suggestion for how to fix it.



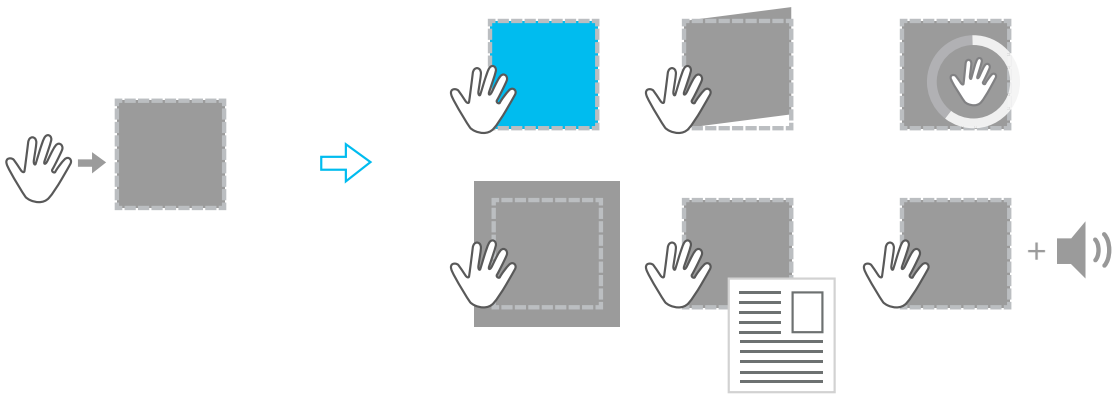
Show cursor visuals if you are tracking a user's hand

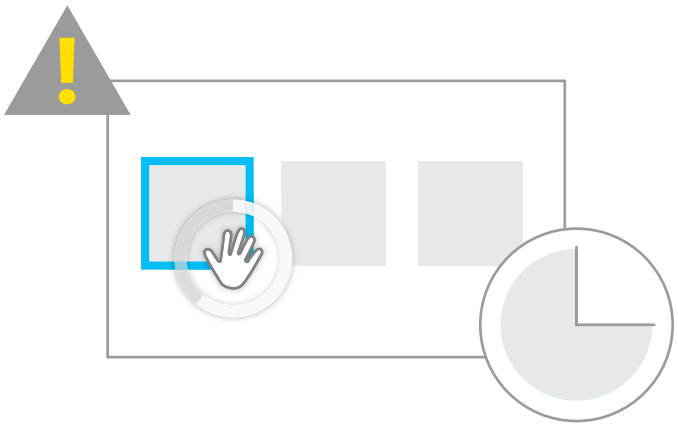
If user is blocked by Kinect error state, notify them and change the visuals. Do not leave UI unchanged. Consider recommending that the user switch to alternative input methods.

Hover and Selection States

For controls that don't navigate, like checkboxes and toggle buttons, have clear visual changes that show state changes

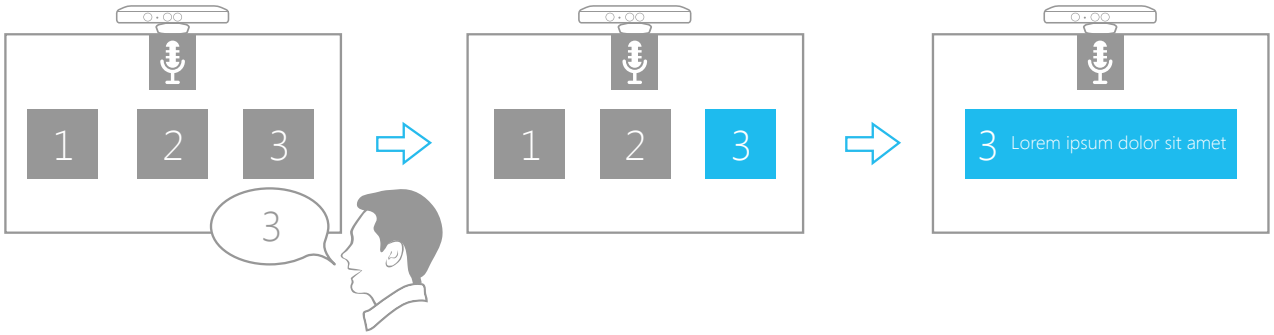
Use color, tilt, orientation, size, or content changes on hover and selection to provide feedback to a user about what items are actionable and what state they are in





Sometimes it may be appropriate to show progress as part of a selection state, but it is important to choose a timing that doesn't frustrate users by forcing interaction to be too slow

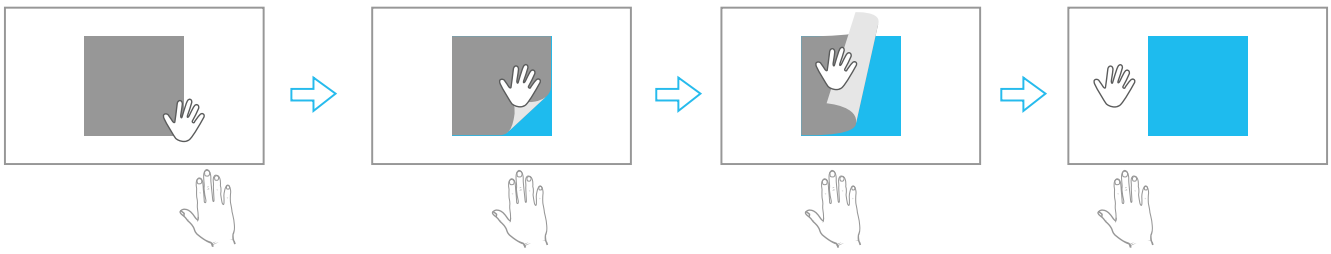
If the user uses voice or gesture for selection, have a visual indication of recognition before changing the state of the application.

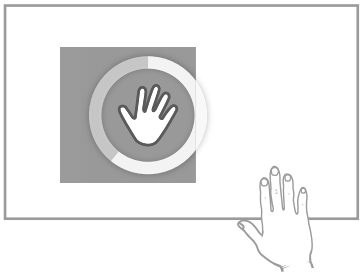


Direct Manipulation

If you are allowing a user to control something with direct manipulation, show the progress in a way that translates to the user's motion

For example, if the user is turning a page with a gesture, show the page pulling up and turning over as the user moves his or her hand horizontally

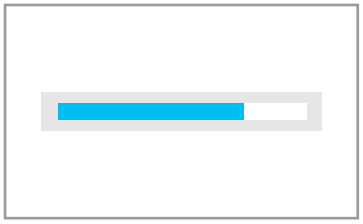




Progress

When choosing timeout length for progress with gestures, consider how long your user can comfortably hold a position and how frequently the user will be repeating the interaction. Avoid forcing the user to wait unnecessarily.

If you are using progress, a timer, or a countdown, use clear visuals to show the entire progression



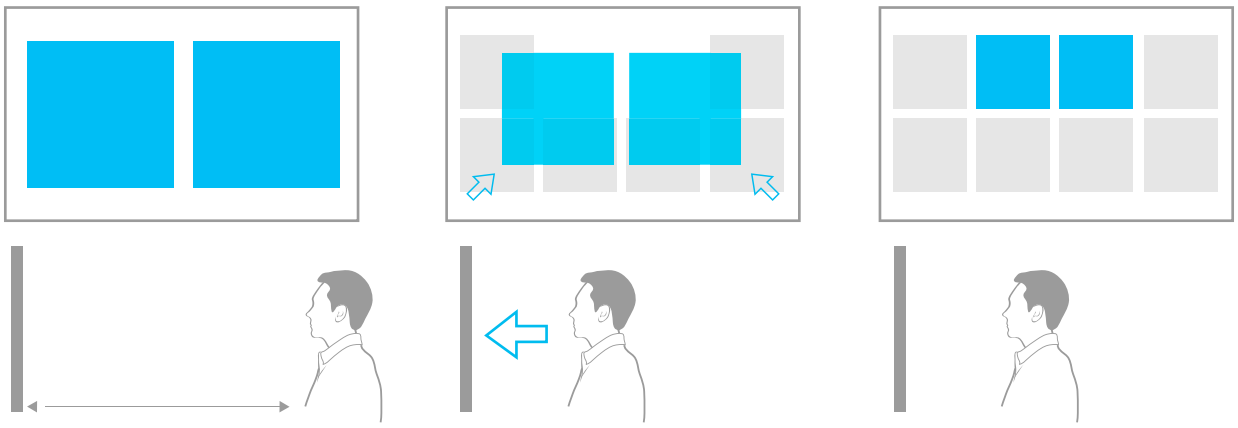
If the progress is towards an important selection, make sure it is prominent and placed where the user is focused

If you expect lag or waiting time in your app, think about using loading or progress UI to avoid letting your app appear frozen

Animation

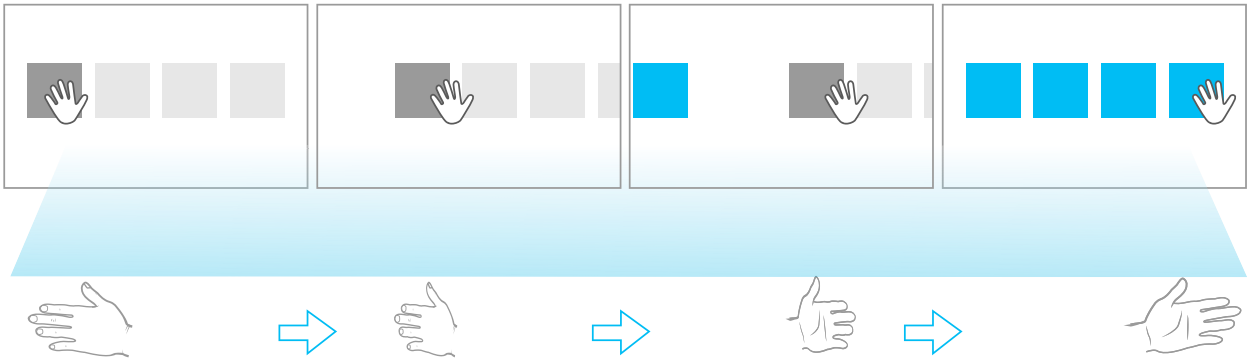
Animation is a great way to help users keep their context

If you are changing the UI based on something that the user may not have knowingly triggered (like distance), try animation as a way to show where the content they were looking at is placed in the new layout

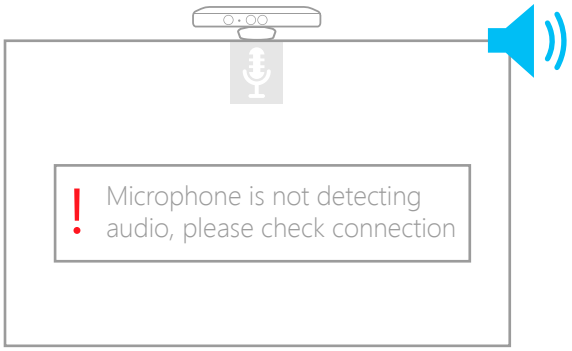


If your user is navigating, use animation to help the user understand the layout

For example, user horizontal movement to show that they've moved horizontally in space in your app's layout. This should also map to any gestures that you are using.



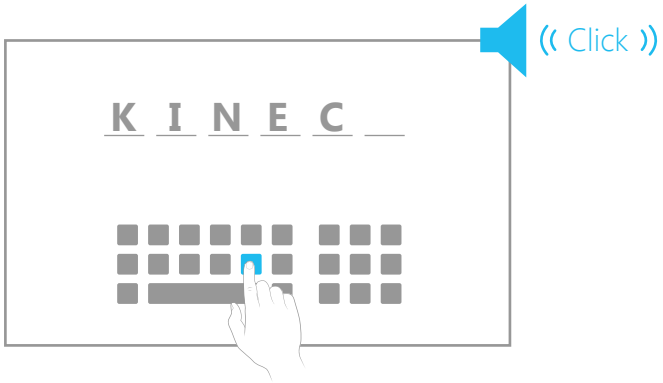
Audio Feedback



Getting user attention

Audio can be a good way to get users' attention if they need to be notified of something
For example: Warnings or alerts

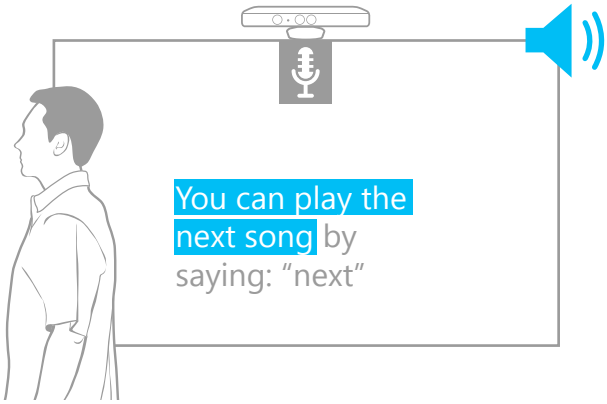
Sound patterns or cues can be used to as a way to communicate a simple message or teach users when to pay attention



Hover and Selection States

Sounds can be a good way to signal to a user that something has changed or happened

Consider using sounds that match the action the user is taking to enforce their feeling of control and familiarity
For example: A typing noise for text input



Audio Feedback Based on User Orientation

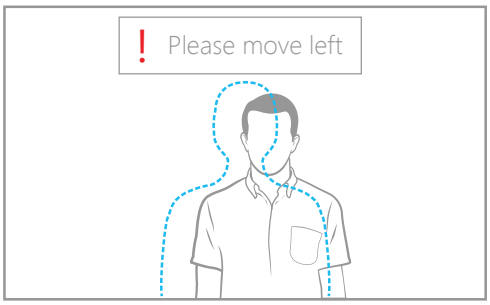
If your user is not facing the screen, or is far away, consider using audio as a way to communicate with them

For example: Options available to them, directions or instructions, alerts

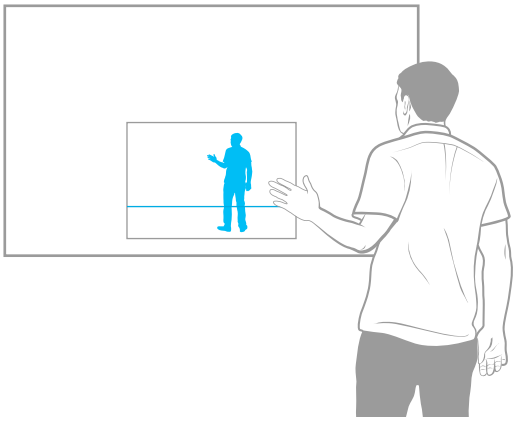
Skeletal Tracking Feedback

Make sure users know whether or not Kinect can see them

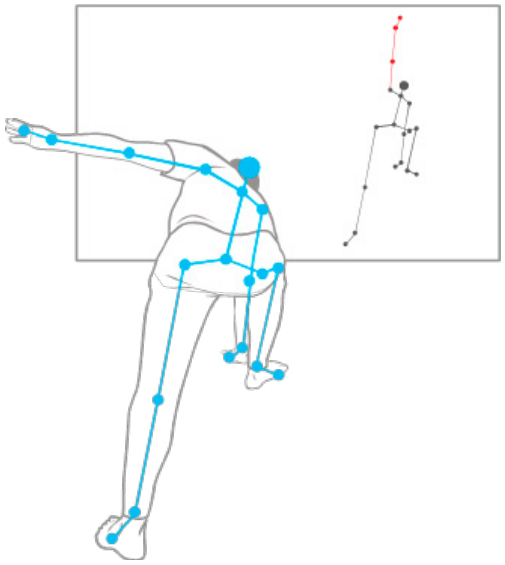
If you need to be tracking a user and cannot "see" or track them, let them know where to stand so that you can. If you lose track of them in the middle of a process, pause it and guide them back to a place where you can track them.



An example of this is in Kinect Sports, where the game tells you if you are getting too close and shows you where to stand for optimal tracking in the beginning.



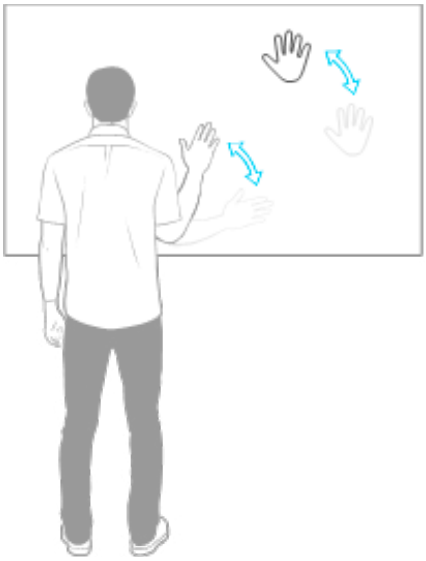
Another way to do this is to show a small depth stream window or visualization to show the user exactly what the Kinect can see at any given time. This may help them understand how to stay within the right range and why things aren't working when they are outside of it.



Lead by Example

If you want your user to copy a specific movement or action, you can do this by showing an avatar or animation, either before you expect to track their movement, or even during. If you are showing an animation during the user's movement, find a way to visually indicate to them whether or not they are doing it correctly.

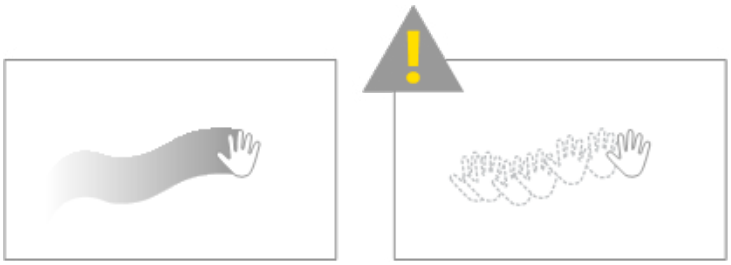
An example of this is in Dance Central, where the dancer on the screen dances correctly and independently of the user, but limbs are highlighted in red when the user makes a mistake.



Mirror

If you are having an avatar mirror the actions of a user keep the following things in mind:

Make sure the movement is real time so the user feels in control



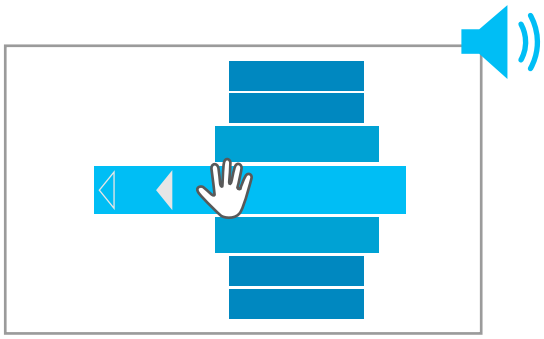
Smooth movements to avoid jittery or not human-like motion and poses

Make other objects in the scene react correctly to collisions with the skeleton and avatar

Combining Feedback

In the physical world we gauge if our actions are effective using all of our senses. Combining different types of feedback can often make for a better experience in a virtual world.

Combining visual and audio inputs can make feedback stronger. A great example of this is the menu in Dance Central. As the user scrolls up and down the menu, the following changes happen:



The currently hovered over list item grows

The currently hovered over list item glows a brighter color

The currently hovered over list item gets a slider visual

The items surrounding the currently hovered over list item are slightly larger and a slightly different color

The sound changes dynamically, in real time based on the position of the user's hand.

The result is a very hands-on feeling experience, where the user can almost feel the effects of their movements.

Basic Interactions

Most magical Kinect experiences will need to be supplemented with basic interactions that also exist in mouse, keyboard and touch interactions, like targeting and selecting buttons, entering text, zooming and panning around a surface, and letting the system know when they want to engage.

Our current knowledge about these interactions and suggestions for how to design them well for use with gesture and voice are described in the following sections.

Designing Strong Inputs

In order to provide a good experience and not frustrate users, your interactions that take voice and gesture as inputs should be:

Low in transactional cost:

- precise
- reliable
- fast
- low effort

Intuitive and have easy mental mapping

Easy to back out of if mistakenly started rather than having to complete the action to cancel

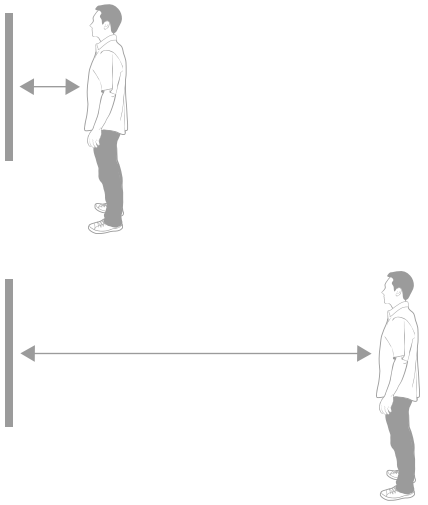
Efficient at a variety of distance ranges. Switching inputs is not ideal

Easy to back out of if mistakenly started rather than having to complete the action to cancel

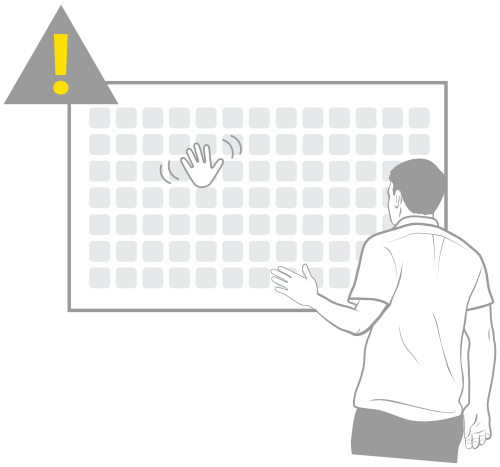
Appropriate for the amount and type of content displayed

Considerate of sociological factors. Users should feel comfortable using the input in their environment

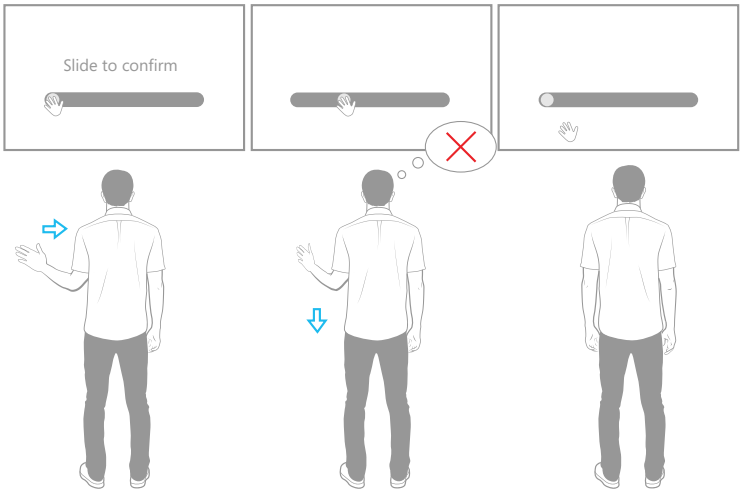
Easy to learn and memorize if needed



Efficient at a variety of distance ranges. Switching inputs is not ideal



Appropriate for the amount and type of content displayed

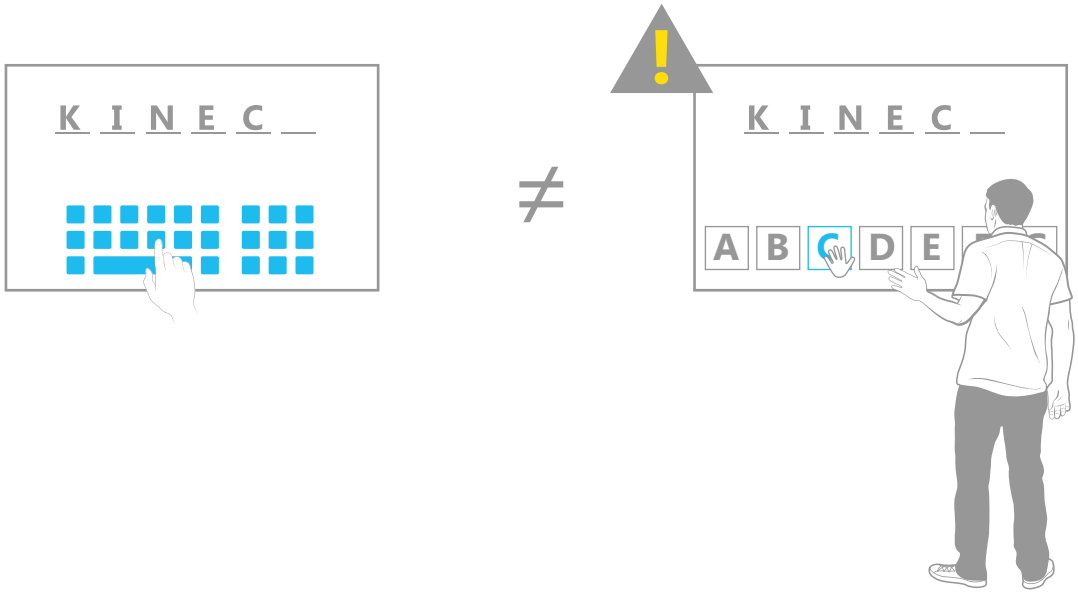


Easy to back out of if mistakenly started rather than having to complete the action to cancel

Choosing the Right Input Mode for your Task

As you design your interactions, keep in mind all and any input modes that are available to you and the pros and cons that each have to offer. If you have one that is particularly good at a given task, consider using it instead of forcing all interactions into one input mode.

For example: For text entry, if you have a keyboard, then have your user use that instead of entering text using gesture.



DO	DON'T
✓ Users can accomplish tasks quickly and intuitively	✗ Use of an input method feels forced, unnatural, awkward or tedious
✓ Input modes are each used for what they are naturally best at	✗ Users are forced to use inappropriate input methods for certain tasks just because they are good for other tasks in the scenario
✓ User setup and position is taken into account so that input modes are only switched when it is convenient and will benefit productivity	

Targeting

In this section we will cover targeting, which is how the user controls the system in order to get to an area they wish to focus on or select. Targeting with a mouse is second nature to most computer users, but as you design your Kinect for Windows application you may want to give users this capability across a range of distances and with different hardware combinations. The information below will help you understand the basics of targeting, various tradeoffs you may have to make based on the scenarios you wish to enable, and some options for making targeting with Kinect as user friendly as possible.

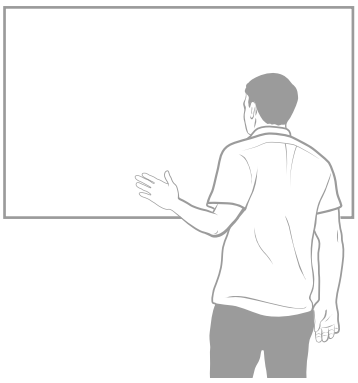
Targeting Basics

For successful targeting, it is best to:

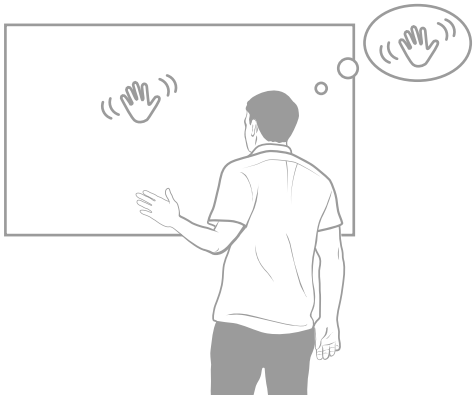
Make it clear what's actionable

Make it easy and comfortable to take action

Make sure visual feedback matches user intent



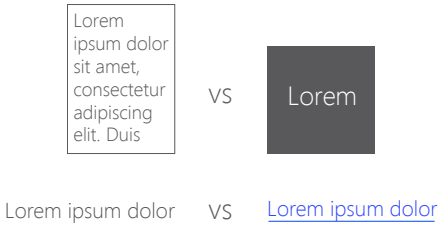
Make it easy and comfortable to take action



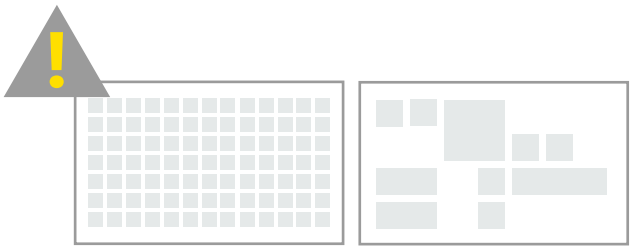
Make sure visual feedback matches user intent

DO	DON'T
✓ Buttons look different from hyperlinks and from static body text	✗ Users must yell commands in a quiet environment
✓ Users feel physically and socially comfortable while targeting	✗ Users must stretch in uncomfortable ways to target parts of the UI
✓ There is a clear visual mapping of the users' movement to the interface	✗ Cursor movement is jittery and disorienting
✓ Controls are easy to target because there the spacing, placement and sizes are appropriate	✗ There is a delay between a user's hand motion and the cursor's movement on the display, causing users to compensate and overshoot their desired target

Do: Make buttons look different from hyperlinks and from static body text

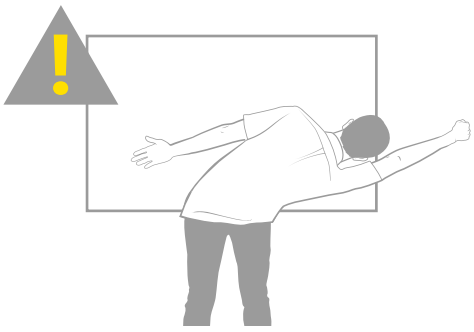


Do: Make it clear what's actionable

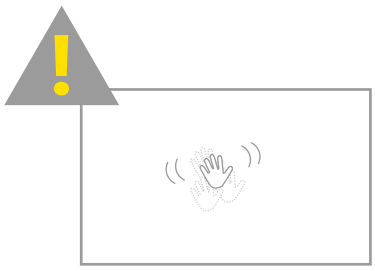


Do: Controls are easy to target because there the spacing, placement and sizes are appropriate

Don't: Make users yell commands in a quiet environment



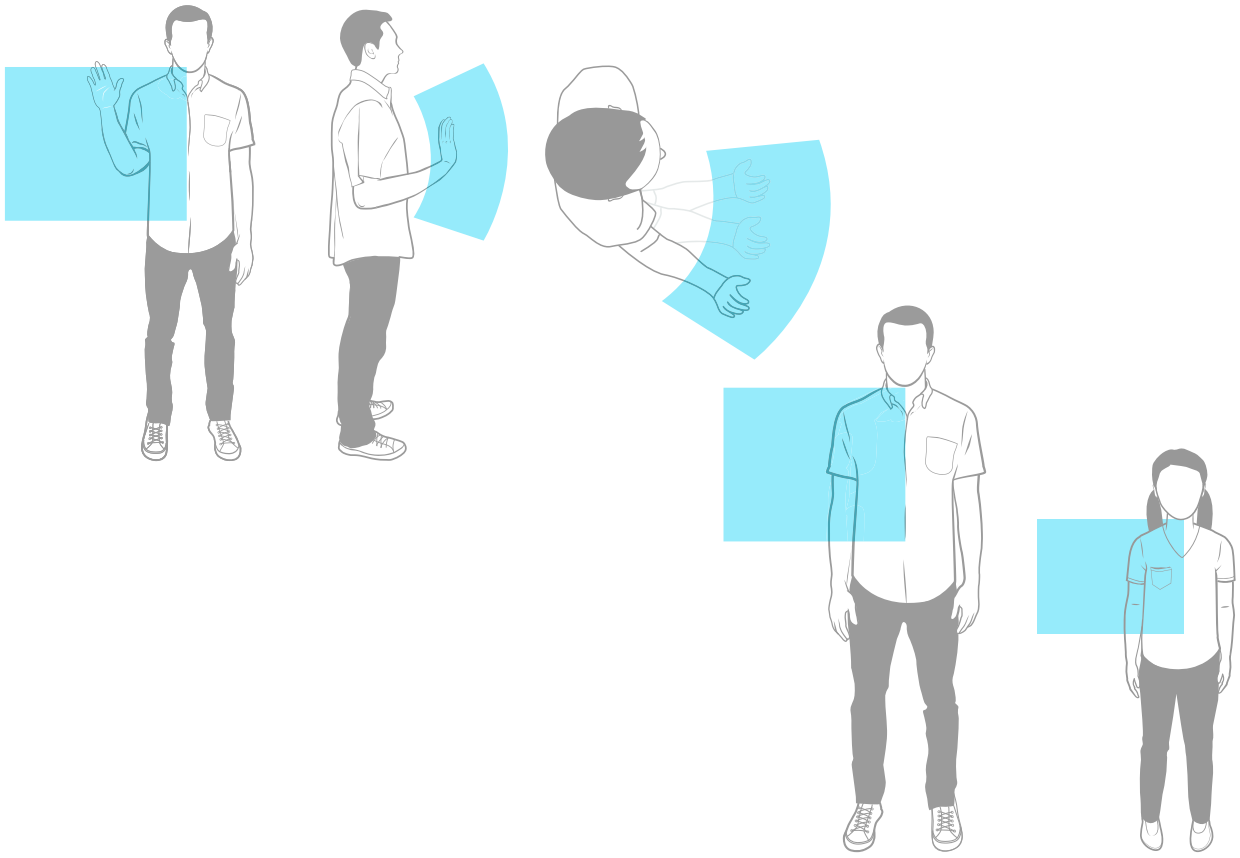
Don't: Make users stretch in uncomfortable ways to target parts of the UI



Don't: Introduce delay between a user's hand motion and the cursor's movement on the display

The Physical Interaction Zone

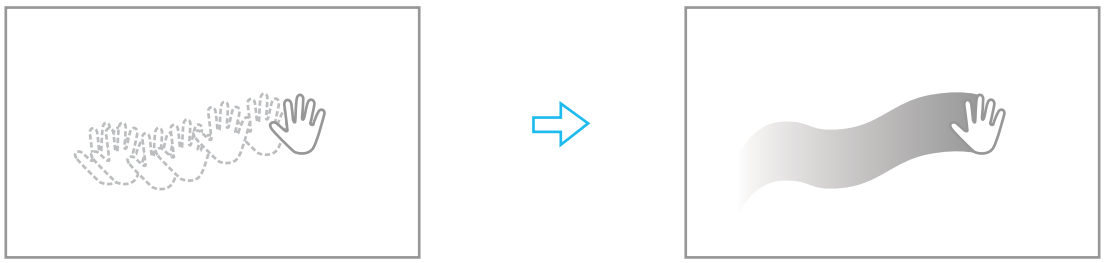
The Physical Interaction Zone, is the area in front of the user where there is a spatial mapping relationship between the user's hand movements in physical space and the cursor movement on the screen. As the user moves his or her hand within the boundaries of the Physical Interaction Zone, the cursor moves within the boundaries of the screen. The Physical Interaction Zone spans from around the head to the navel and is centered on the range of motion of the hand on the left and right sides. The depth of the Physical Interaction Zone is curved because of the range of motion of the hands and arms are arc-shaped. The Physical Interaction Zone is relative to the size and orientation of the user.



Cursor Implementation

The most common way of targeting, and the way targeting is done for Kinect on Xbox360 is with a cursor visual that is controlled by hand movement. The simplest way of implementing a cursor is to define a Physical Interaction Zone and just do a direct mapping of the horizontal and vertical position of the hand in the Zone to the screen. This is how the Xbox360 cursor is implemented.

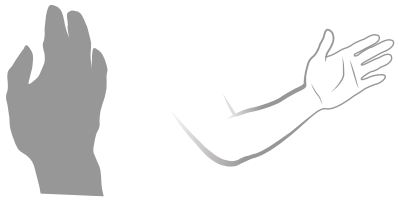
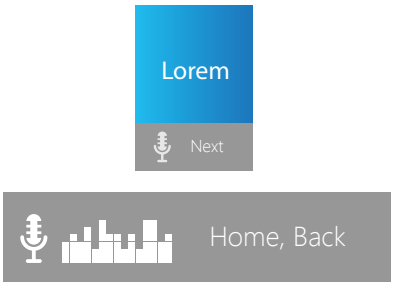
Some may choose to refine their cursor implementation by adding a filter to reduce jittery and jumpy behavior that may be returned by the skeleton data. A simple filter example would be to use the average or median location over a number of frames to weed out unlikely or unnatural large movements



One side effect of using filters is that it often increases lag. Lag can greatly compromise the experience, making it feel slow and unresponsive. Finding a good balance between smoothness and lag can be tough, and may require extra time for iteration.

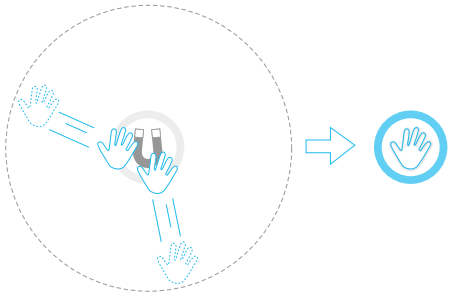
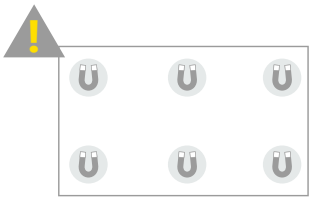
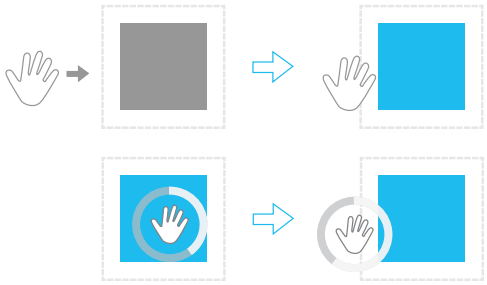
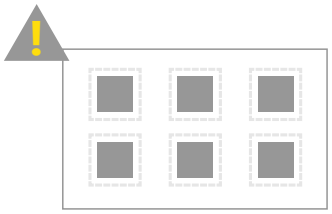
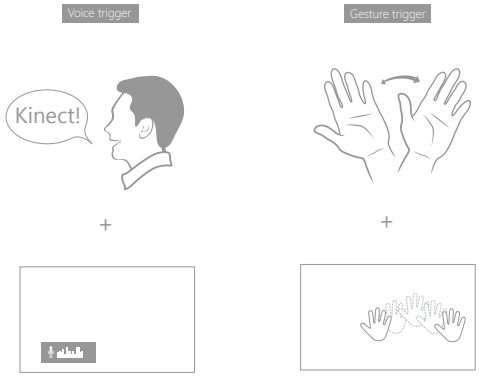
Cursor Design

While there are many ways to design a cursor, here are a few that we've tried and tested and know are easy for users to understand.

	DESCRIPTION	EXAMPLE VISUALS
Cursor Symbol	Static symbolic representation that simply maps the user's intended movement to 2D space. If using a hand symbol, consider changing it based on hand state or if user is using left or right hand.	
Depth 3D representation	Depth representation of the user's hand that shows how the user's fingers, hand and sometimes lower arm position is being mapped to a UI location. Helps give the user feedback on orientation and depth in the z-axis	
VUI Voice User Interface	Text in the UI that indicates what the user can say as a spoken command. VUI should be differentiated in a clear way to help user understanding.	

More Targeting Interaction Options

Targeting from a distance can be difficult because of the decreased level of precision and direct control. Here are some ways that you can make your UI more forgiving and clear.

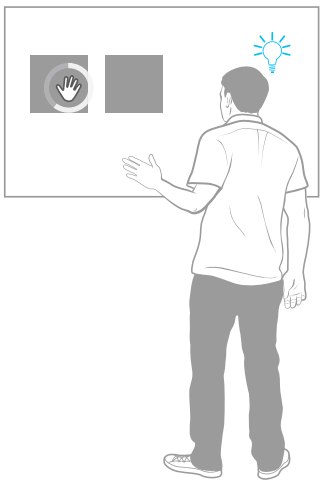
	DESCRIPTION	HELPFUL NOTES
Magnetism	Snaps a cursor to the magnetic control if it is within a certain radius of the control. Effectively makes an actionable area much larger than the graphic representation of the area, which allows for high precision while requiring less effort on the part of the user. 	* This should be reserved for a very small number of controls that are relatively spaced out. Too much magnetism on a screen is frustrating and the user loses perceived control of the cursor as it reacts to magnetism and no longer follows the user's motion. 
Increased Collision Volume	Increases the range beyond the visuals for which a control will respond. For example, a button may show its hover state if the cursor is near but not directly over the button. Can also be turned on once a timer is started so that the user's hand can drift outside of the button slightly without disabling the timer. 	* Can be confusing if used too frequently or close together. User should not feel that they are constantly triggering controls that they are not targeting, just by moving their cursor around the UI. 
Trigger + Mode	Use voice or a separate gesture to get into "targeting mode" 	* You should only show the cursor when in targeting mode. * Make it clear how to get into this mode. * Controls on screen should not appear targetable unless you are in targeting mode.

Selection

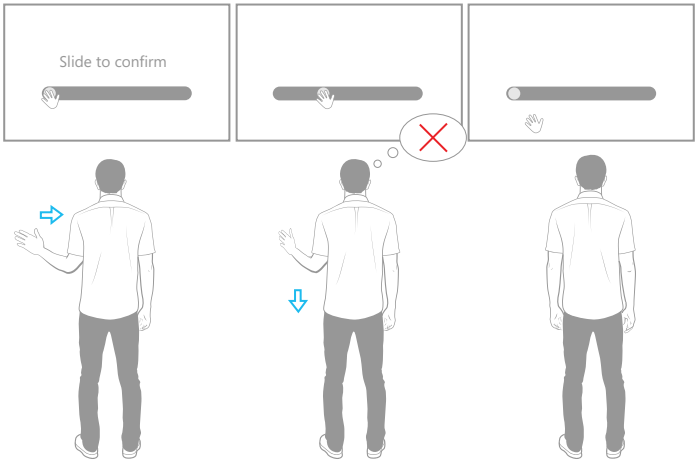
Selection is how a user makes a deliberate choice to trigger an action, like left clicking with a mouse. Since selection can often trigger important changes, it is important v to really be confident about your users' intent. For example, if there is a "purchase" button on the screen, you do not want to make it easy for your user to mistakenly select it. Gesture introduces some amount of uncertainty to selection that must be mitigated and managed through interaction design.

Selection Basics

Selection should be reliable and deliberately controllable.



Users are aware that an item is being selected

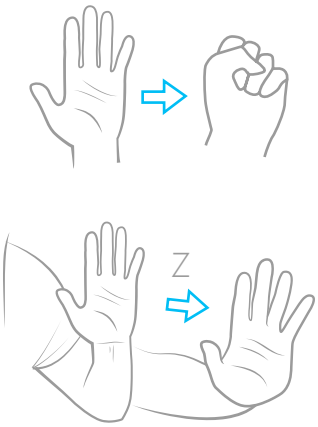


Users can cancel out of a mistaken selection before it is made

DO	DON'T
✓ Selections are triggered with high confidence in the users' intent	✗ Controls are frequently selected by mistake as the user interacts with the system
✓ Users are aware that an item is being selected	
✓ Users can cancel out of a mistaken selection before it is made	

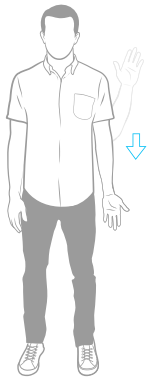
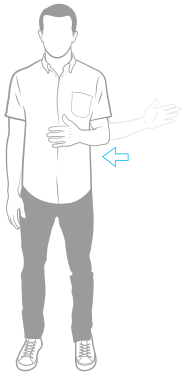
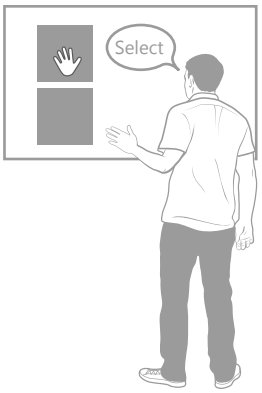
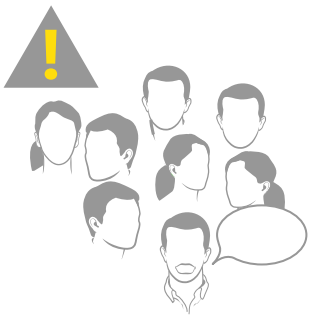
Selection UI Design

The UI you choose to use for selection will most likely be based on the UI you used for targeting, but here are some options that we suggest.

	DESCRIPTION	EXAMPLE VISUALS
Progress Cursor	Progress UI that fills up as a cursor hovers over a target for a specified amount of time. It is important to find a good balance between reducing unintentional selections by making the timeout longer, and not inducing fatigue and frustrating the user by making interaction slow. Changing the progress animation rate or start time can significantly impact perceived performance without affecting actual selection times.	
Change hand shape orientation, or z position	Make a fist, change hand orientation, or push in to select. Keep in mind that these interactions are particularly challenging to make reliable because the cursor will unintentionally shift in X and Y directions as the user presses in the Z direction. Providing good feedback, especially for z-based selection is critical.	
VUI	Distinguish text clearly that can be read as a command.	

More Selection Interaction Options

Selecting via gesture can be difficult because it often requires the user to hold their hand still for a certain amount of time. Below are some options that can help make selection feel more like a deliberate action.

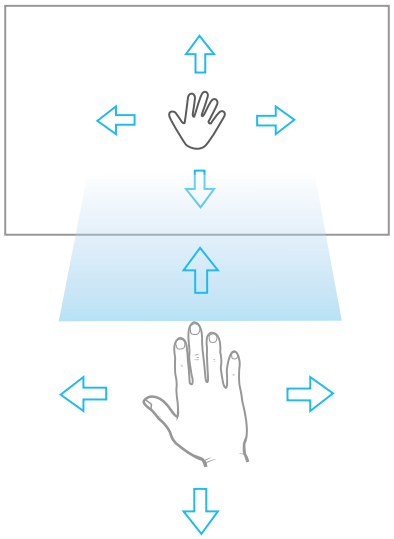
	DESCRIPTION	HELPFUL NOTES	
Dynamic Gesture	Shows directly-mapped progress as user goes through with selection.	✱ The time and movement required for this type of selection allows users to get out of a selection that they mistakenly have started and to be more deliberate about a selection that they want to make. Make sure to give clear feedback so users understand how to back out of a selection if necessary. ✱ It helps if the axis of the selection gesture is not the same as the targeting axis. For example, if a user is navigating in a vertical list of controls, then the selecting gesture should be horizontal.	IF TARGETING IS:  THEN SELECTION SHOULD BE: 
Increased Collision Volume	Use a voice command to trigger selection after using gesture to target. **For example**: after hovering over a button with a cursor, say, "Select." 	✱ This can be a way to make faster selections since there is no timer and you can be fairly confident about the user's intent. This would not work as well in louder environments or in places where it is socially unacceptable to be speaking loudly.	

Panning

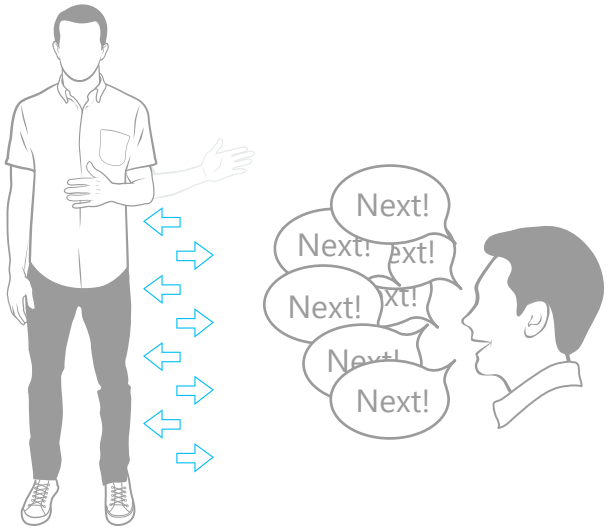
Panning allows users to navigate up, down, left, right and sometimes freely within a canvas, like dragging your finger on a touchscreen. It can be used for navigation on a surface or page or navigating between consecutive screens or views. While voice tends to not offer a strong solution for panning, gesture offers some interesting options for panning at a distance. Below are some examples of how you can enable panning in your applications.

Panning Basics

Panning changes the location of the user in the UI. The larger the canvas or structure of the application, the more the user will have to pan. Also, since panning often requires direct manipulation, responsiveness and accuracy are very important. Keep these things in mind as you design, so that your user can get where they need to go quickly, accurately and without fatigue.



Panning gestures semantically and directionally match the resulting movement

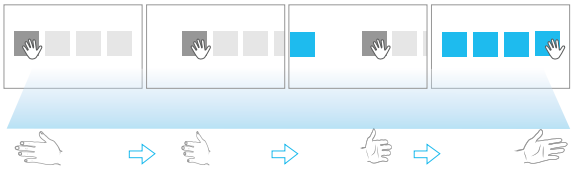
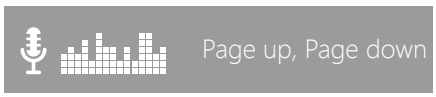


Don't make users must perform a lot of repetitive gestures or voice commands to pan longer distances

DO	DON'T
✓ Users can pan large distances in a large canvas without too much repetition or exertion.	✗ Users must perform a lot of repetitive gestures or voice commands to pan longer distances.
✓ Users have control over where they are panning and can easily target specific areas.	✗ UI jumps around and feels disorienting , making it difficult for users to find specific content.
✓ Panning gestures semantically and directionally match the resulting movement.	

Panning UI Design

Panning feedback comes from the UI view changing to accommodate the desired movement paired with common control or gesture recognition. Below are some simple UI options that are familiar to users. As mentioned above, voice is an option, but may provide a tedious panning experience.

	DESCRIPTION	EXAMPLE VISUALS
Target Zone and Swipe	Target a zone on the screen and then swipe. This is how navigation is done in Xbox360.	
Have scrollbar-like UI target and select, or target and hold	Directional arrows and/or a scrollbar that can be targeted and selected.	
VUI	Voice command options that allow users to pan, such as "page up," "page down," "right," "left."	

More Panning Interaction Options

Here are some approaches you can take to help users get where they want faster and more accurately.

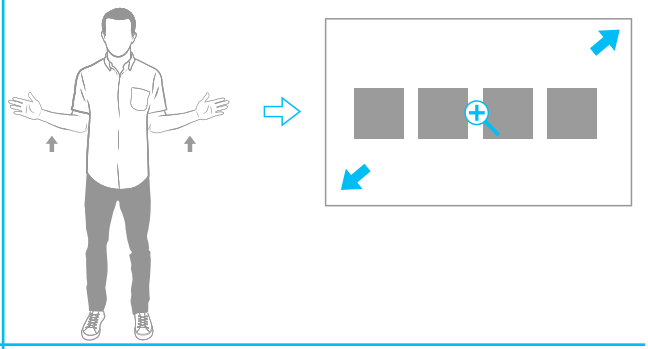
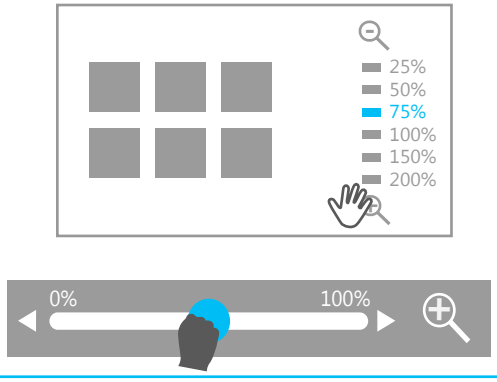
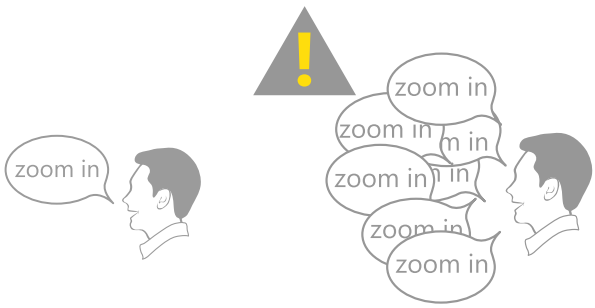
	DESCRIPTION	HELPFUL NOTES
Pan with acceleration	As user holds hand over a zone or scrolling UI, scroll speed increases. Alternatively, control the speed of the motion based on how far the user's hand is reaching.	* Increasing acceleration is great for long scrolls but makes it hard to make very targeted scrolls/pans to a specific location * Controlling the speed of the motion based on the reach of the hand allows users to slow down the motion without stopping.
Canvas target, select, target	Have a gesture or other command that is used to signify that the canvas should move to remain under the user's hand. 	* It is important to avoid false positives here. It could be very disorienting if the canvas was selected by mistake. * Consider using momentum so that canvas continues moving when selection is released to give it a smoother feel.
Set Increment	Design a simple, single directional gesture that triggers panning in a direction with the same increment panned/scrolled for every gesture.	* This can be tedious and frustrating for long scrolls * This works well for in-app navigation where there's only one main type of increment anyway (next page, previous page)

Zooming (z-axis panning)

Zooming is very similar to panning, but instead of moving vertically or horizontally you are moving towards and away from the canvas. Many users are used to using a zoom control with mouse and keyboard or pinching to zoom on a touch screen. Zooming with Kinect can be especially challenging because users are used to the level of control that they get with touch zoom and it is much harder to be precise about start and end points and distance when you are not directly touching the surface. Below are some simple solutions for zooming that will still be familiar to most users.

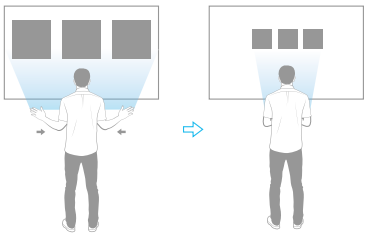
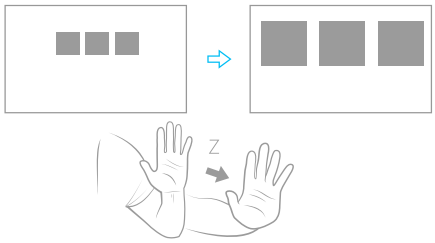
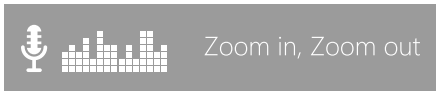
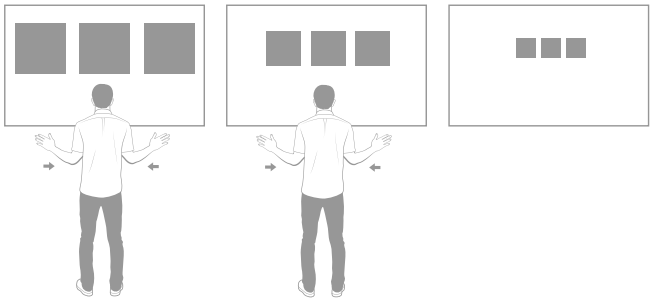
Zooming UI Design

Panning feedback comes from the UI view changing to accommodate the desired movement paired with common control or gesture recognition. Below are some simple UI options that are familiar to users. As mentioned above, voice is an option, but may provide a tedious panning experience.

	DESCRIPTION	EXAMPLE VISUALS
Trigger a Zoom Mode	If zoom is triggered, for example, by a user holding both hands up, show UI that indicates that zoom has been initiated. This UI could represent the hands or be more abstract or symbolic, like arrows.	
Zoom control UI (target and select to move, or target and hold)	Zoom UI is often similar to a slider that goes from 0 to 100%. It may be useful to allow users to grab and drag the slider, as well as press and hold on either side.	
VUI	Allow users to jump to different zoom increments with commands like “zoom 100 percent,” “zoom in” or “zoom out.” Avoid forcing users to do too much repetition of voice commands.	

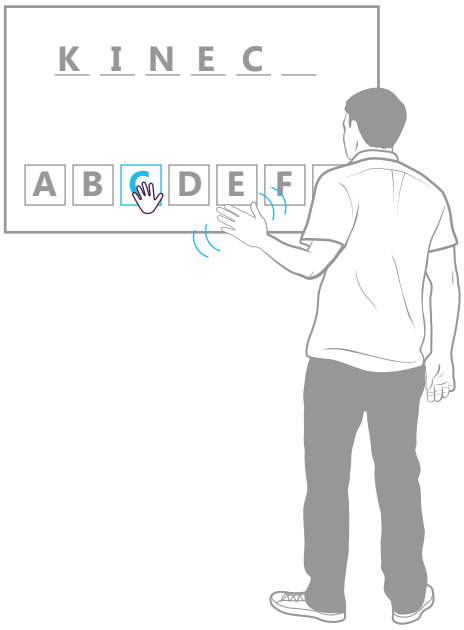
More Zooming Interaction Options

The goal of zooming is to manipulate an object on the screen and see the results. Here are some ways to map user actions to zoom responses. As with anything that requires direct manipulation, try to avoid lag and keep the UI responsive as possible.

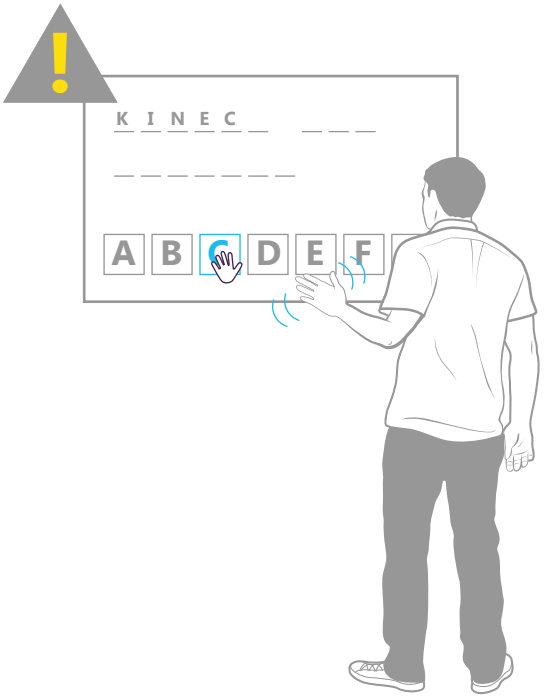
	DESCRIPTION	HELPFUL NOTES
Two Handed Zoom	Look for user to hold up both hands, and then base percent of zoom on percent of change in distance of hands towards or away from each other. 	* This may feel most intuitive and familiar to users since it is similar to the touch interaction. Implementing it well may be challenging. * You will need to define the start and end of the gesture accurately to make the zoom level stick at the desired location.
Z Space Zoom	Use a literal translation between user's hand and object by zooming with hand moving in z space. Increase zoom as users hand pulls in towards body, and decrease as the hand pushes out. 	* Z space is typically harder to make accurate, so this may be a more challenging, albeit intriguing approach.
Use Trigger + Zoom Gesture	Have an allowance for user to trigger zoom with voice or a distinct gesture and then zoom with another gesture. 	* It is important to avoid false positives with this method because it could be very disorienting if zoom is triggered by mistake.
Set Increment	Have an allowance for user to trigger zoom with voice or a distinct gesture and then zoom with another gesture. 	* This can be tedious and frustrating for big zooms, but may work well for semantic zooms.

Text Entry Basics

Writing content with gesture or voice is very difficult, slow and unreliable and should be avoided. Searching or filtering can be done successfully if the user must only select a few letters. Keep in mind that text entry implementation with gesture is usually a series of targeting and selecting actions, which can get frustrating and tiring if it must be done multiple times in sequence.



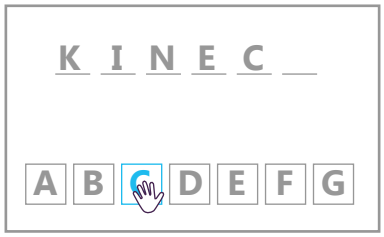
DO: Text entry for search or filter through a small set where only a few letters must be entered



DON'T: Have long phrase dictation or random voice input

DO	DON'T
✓ Text entry for search or filter through a small set where only a few letters must be entered	✗ Long text entry with gesture that imitates a keyboard experience
✓ Voice text entry with a small number of recognized words	✗ Voice text entry with letters (sounds are too similar: "E," "P," "G")
	✗ Long phrase dictation or random voice input

Text Entry UI Design

	DESCRIPTION	EXAMPLE VISUALS
Virtual Keyboard	A solution used to allow for text entry via targeting and selecting from a list of letters.	

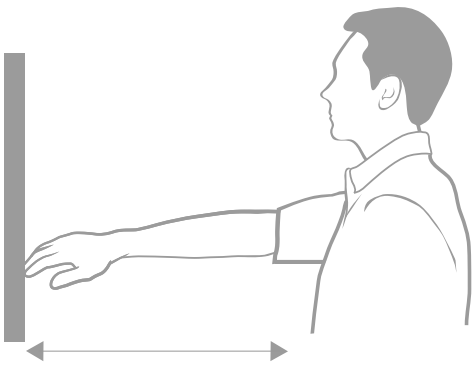
Distance-Dependent Interactions

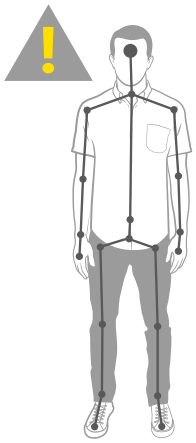
An exciting and new opportunity that Kinect adds is that it opens up a whole new distance range of interaction for app developers and users, removing the requirement that a user be close enough to directly touch a computer in order to interact with it. This also introduces another set of considerations for designing interactions and interfaces. This section will help make you aware of steps you can take to make your interface more usable at any distance.

Spectrum of Interaction

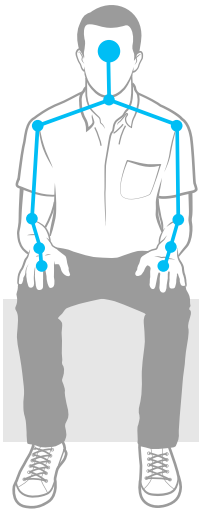
Tactile (0.0-0.4 Meters): Tactile is the range at which a user could use a mouse, keyboard or touch screen, thus by definition, they must be no further than an arm's length away. Since the user is close to the screen, this range can have the highest level of detail for visuals and controls.

If your user is in this range, mouse, keyboard and touch may be more appropriate input methods than voice or gesture. Depth data is limited to beyond 40 cm in Near Range, 80 cm in Default Range, so depth or skeleton data are not available at this distance.

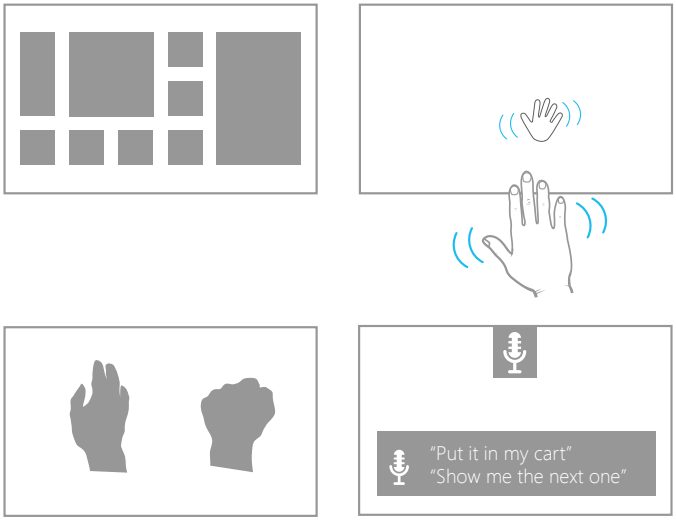




Near (0.8-2.0 Meters):
Near range is where a full skeleton may be partially obscured or cut off, but this is a good distance for seated mode.

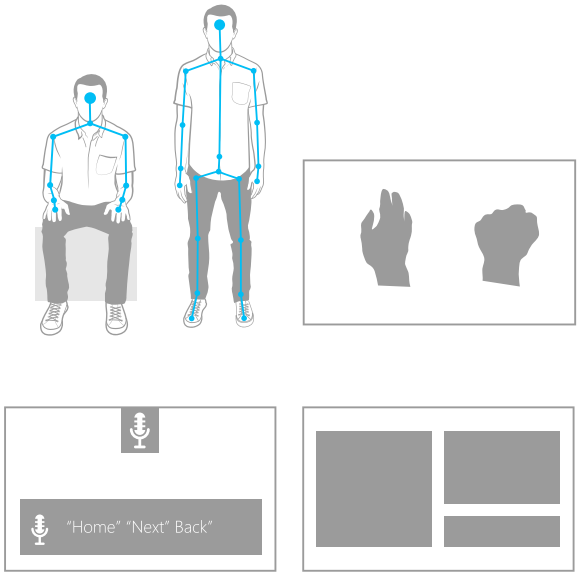


Since the user is still near, visuals can still be fairly detailed, longer phrases can be used for speech, and finer gestures and depth data can be used. This is also a good range to work in if you plan on doing object or symbol recognition.



Far(2.0-4.0 Meters):
In far range, the full skeleton should be visible. Seated mode is also available.

Coarser gestures and shorter commands work best in this range. UI must be large enough to be visible from far away.

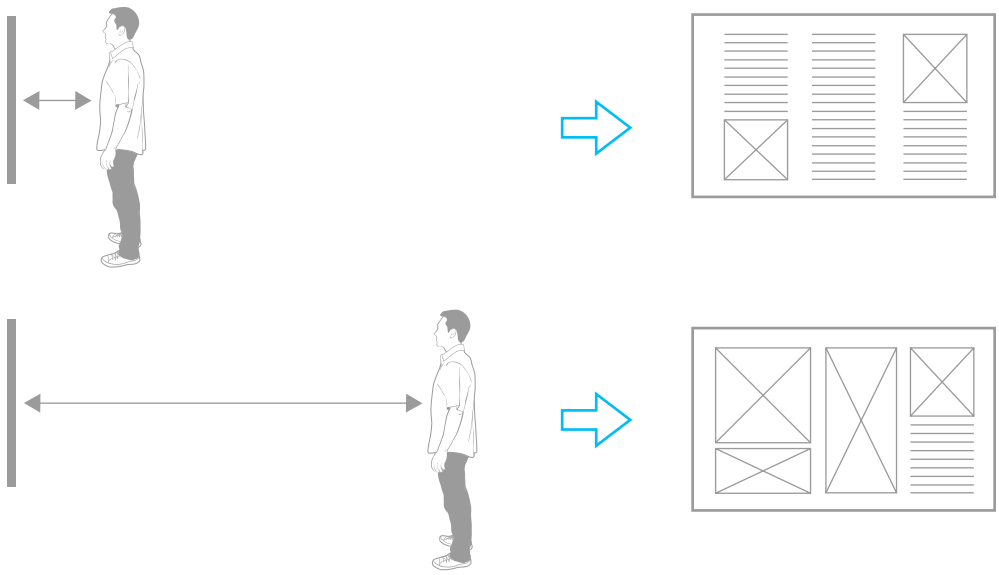


Out of Range (>4 Meters):
Most Kinect interactions are not feasible at this range. Your UI should focus on informing users that there is an interesting interaction available and enticing them to move closer. Visuals must be very large and simple and in some cases audio may be used to get users' attention.

Content, Text and Target Sizes

As the user gets further away from the screen, the text and target sizes should adjust to account for how the distance may affect what the user can see, as well as imprecision and decreasing detail in hand tracking and gestures.

As distance increases, consider increasing the graphic content to text content ratio to maintain user engagement and reduce visual strain. As the UI adjusts to the users' distance, use transitions to help them keep their context and understand how their movements are affecting what they see.

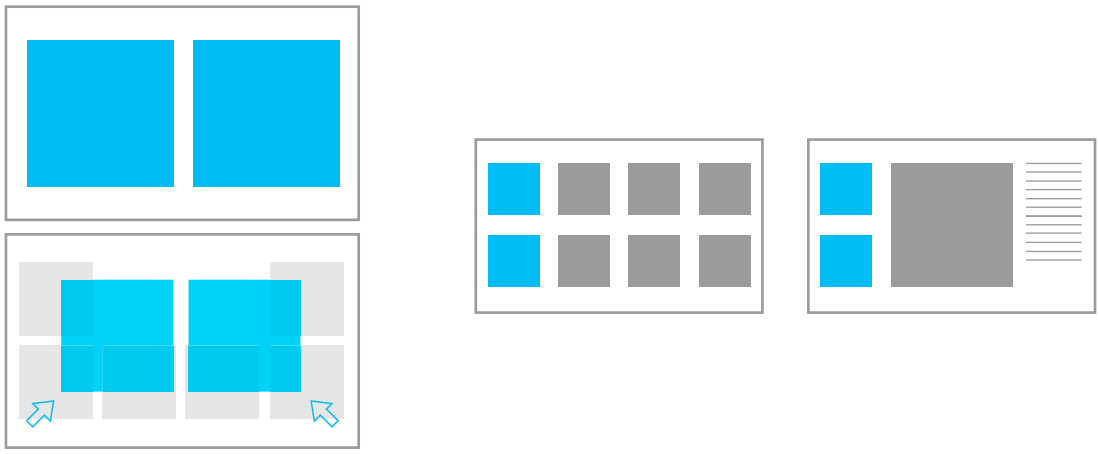


Transitions should:

Be smooth

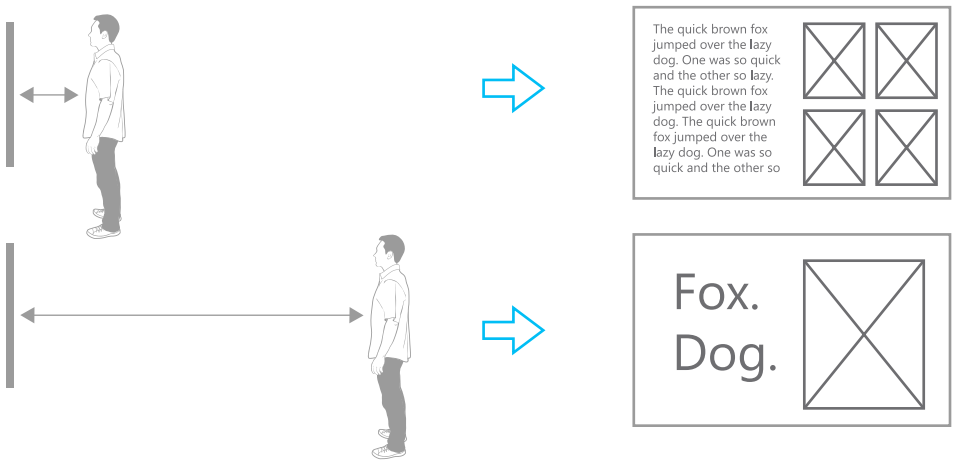
Consider using anchor points or unchanging sections to orient the user and provide consistency

Keep general location of actionable items consistent

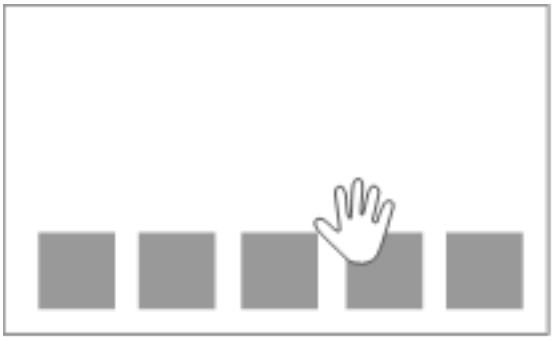


Actionable Item Placement and Numbers

When using gesture, distance as well as screen size should factor in to determining how many actionable items you can have on the screen and where they should be placed. Here are some things to keep in mind:



Longer distances mean text has to be larger and gestures less detailed so this will most likely lead to fewer gestures and voice commands in the far range.



In general, grouping actionable items in an easy to reach area may help users have faster interactions

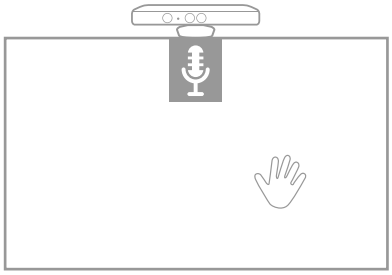
Multimodality

Much like how in human interactions we often use both voice and gestures to communicate with each other, sometimes the most natural way to interact with an application may use multiple input methods. Of course, many options and combinations can put you at risk for a confusing experience. This section will explore asynchronous and synchronous interactions and how to make them clear and usable.

Asynchronous Interactions

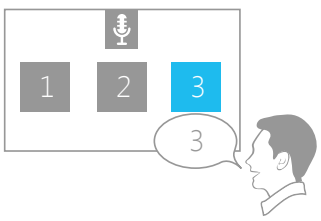
Asynchronous Interactions are ones in which multiple input modes are available but only one is used per action. For example, your application may allow users to use their voice to trigger some commands, and gesture to navigate, or it may use keyboard for text input in the tactile range and skeleton tracking and gestures at a distance. Here are some things to remember:

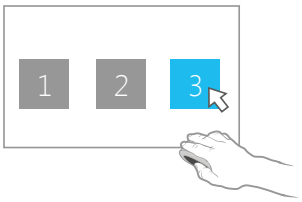
Provide visual and/or audio cues to the user about what input methods are available



Whenever possible, have a back-up input method that can be used to complete any given task

For example: Mouse or voice for selection of a button





Synchronous Interactions

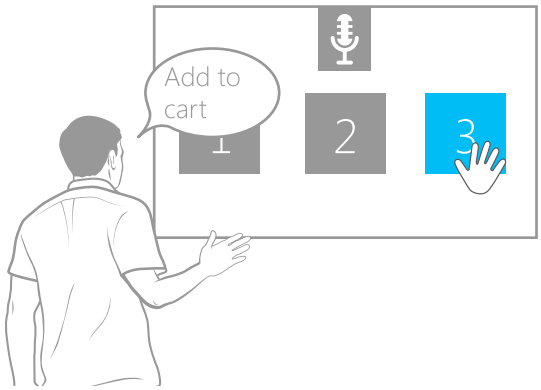
Synchronous interactions are ones in which multiple input methods are used in sequence to complete a single action.

Some examples are:

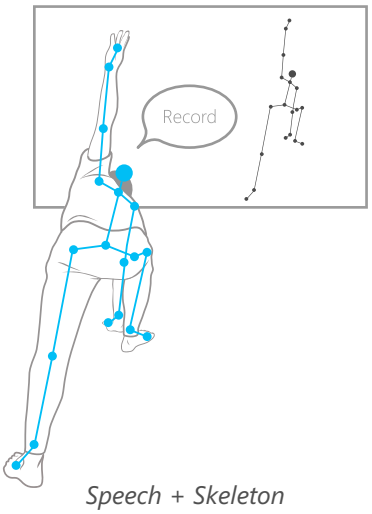
Speech + gesture
Point to product, say “add to cart”

Speech + touch
Tap and hold photo, say “send”

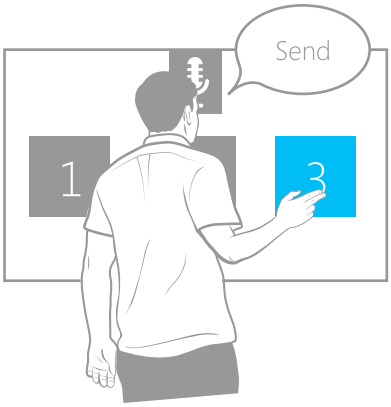
Speech + Skeleton
Say “record” and make a movement



Speech+Gesture



Speech + Skeleton
Say "record" and make a movement



Speech + touch
Tap and hold photo, say "send"

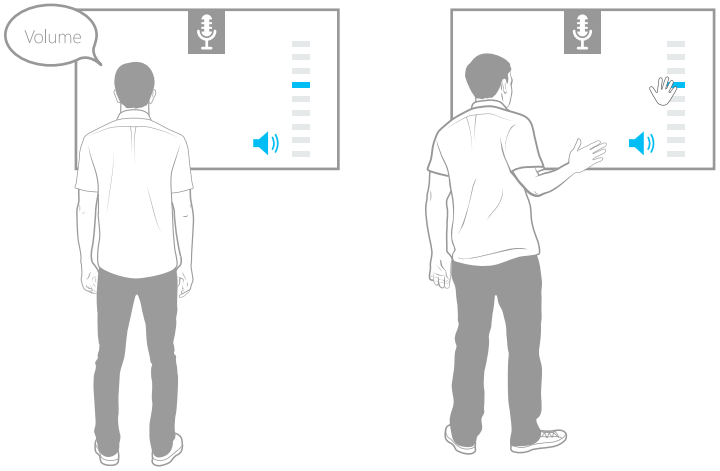
Some reasons to use Synchronous Interactions

Reduce the number of steps for a more complex action

For example: Usually sending a photo takes two steps from the same input method : click photo, click send. Multimodality allows this to become one action with two input methods

Make triggering controls possible without moving your body

For example: Usually sending a photo takes two steps from the same input method : click photo, click send. Multimodality allows this to become one action with two input methods



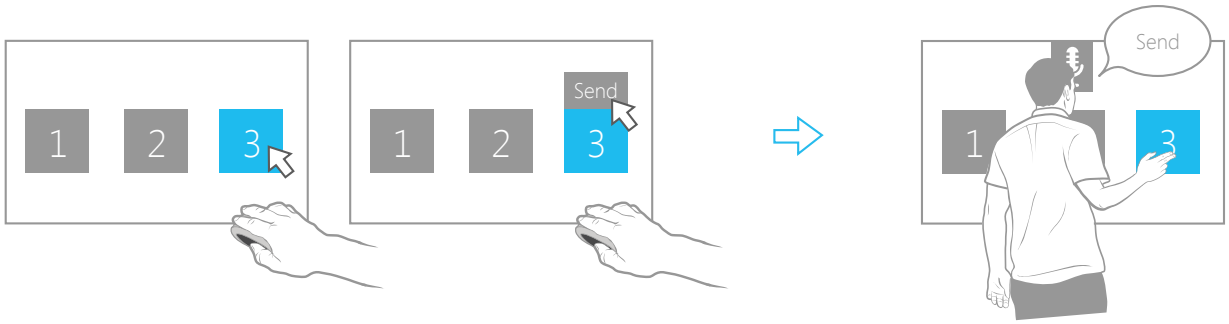
Reduce the number of steps for a more complex action

Increase confidence

For example: If a user has to do two different things to perform a command, it is more likely that the command is executed purposefully.

Make the best use of each input

For example: Using voice to put the system into a volume setting mode but using gesture to give the analog input for the volume level



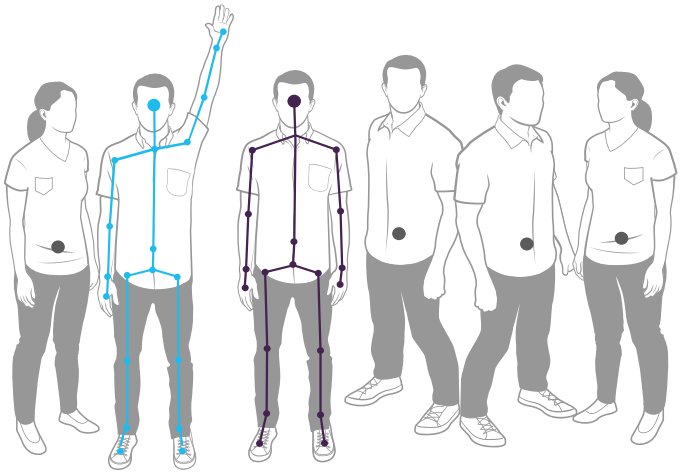
Reduce the number of steps for a more complex action

Multiple Users

Another magical feature that Kinect offers is the ability to track multiple users. This opens a door for some interesting collaborative interactions, as well as some new challenges around control and inputs. This section covers what we know so far about dealing with multiple users.

Tracking Abilities for Multiple Users

Kinect can be aware of up to six people, and can track two of them in detail. For the two skeletons that are tracked, Kinect returns full joint information. For the other skeletons, Kinect still returns useful information: a "player cutout" specifying the outline of the player, and a center position. The application can choose which of the six players should receive full skeleton tracking.



Multiple Users and Distance

As you design experiences and build expectations, keep in mind how distance affects how many people can physically be in view.

You can expect the following number of people to be able to comfortably interact with or be seen by your application in each distance range:

	TACTILE	NEAR	FAR	OUT OF RANGE
Number of People	0	1-2	1-6	0

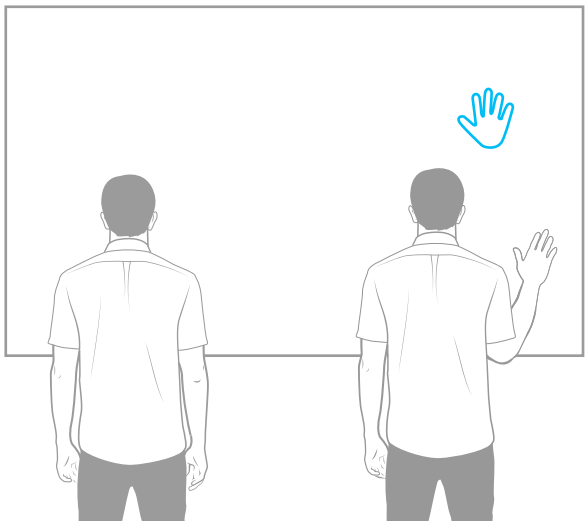
Multiple Users, Multimodality and Collaboration

Enabling multimodality and multiple users means that you may have many combinations of users at different distances using different modalities. Depending on your model for collaboration, there are some important considerations to be made for different ways of handling this.

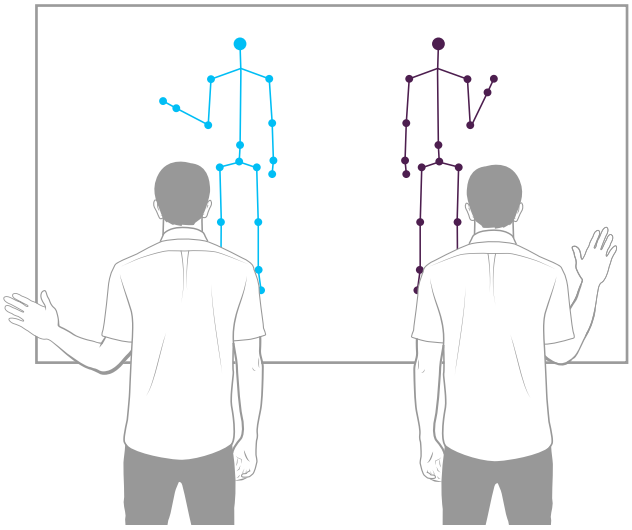
Collaborative Interactions

In collaborative interactions, both users are interacting with the same screen space and inputs. This means that any action taken by one user will affect the state of the application for both users. There are two options for balancing control in this case:

Single Driver Model: This model assigns one of the users as the "driver" at any given time and only registers actions taken by that user. The driver role can be selected or transferred in a number of ways, including choosing the first user to engage, or the user that is closer to the sensor. This is one way to avoid conflicting inputs. This model is usually indicated by having visuals that show which person is being tracked and only having one cursor on the screen at any time.



Single Driver Model



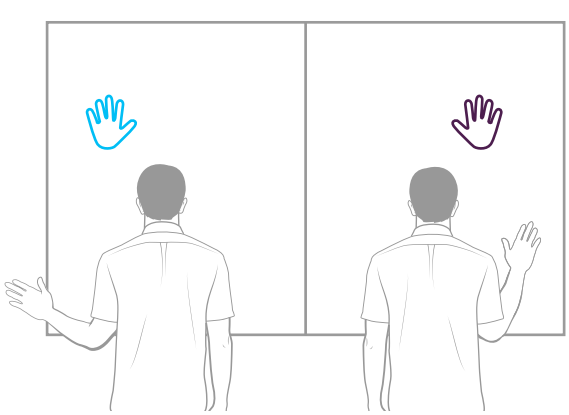
Equal Participation Model

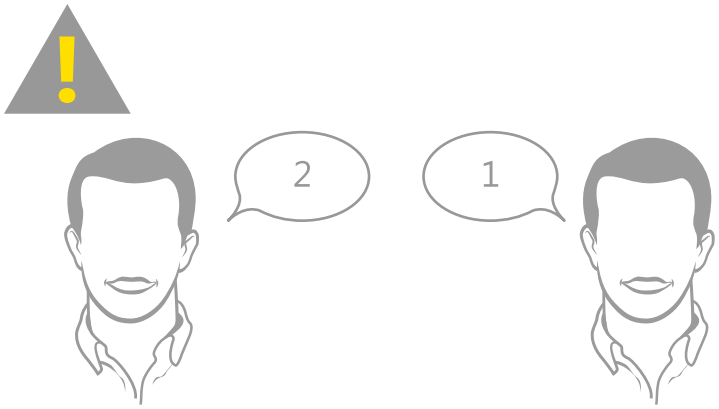
Non-Collaborative Interactions

Non-collaborative interactions give each user their own sub experience within the interface. This can be handled with screen partitioning or sectioning. In general this experience should behave much like a single user experience; however, an extra challenge presented is correctly mapping the user's speech and actions to the corresponding interface. Voice commands may prove to be a challenge. Gestures and movement should have separate visual feedback per user.

Equal Participation Model

This model takes inputs from both users, often simply in the order that they are given. This can be a very complex model for navigating or basic interactions but works very well for scenarios where each user has a unique experience to drive, as in avateering, or games.





Conclusion

Interactions and interfaces built with Kinect can enable magical experiences for your users. The magic and delight of Kinect comes when the user knows how to use the interface and feels natural doing it. Your development of touch free, natural UI with Kinect is the beginning of a revolution that will shape the way we experience and interact with software applications. Someday you will not simply remember how the Kinect revolution started, but how your application helped shape it.