



 /IntelRealSense
 /IntelRealSense
intel.com/RealSense
intel.com/RealSense/SDK

SDK Design Guidelines

version 2

Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

*Other names and brands may be claimed as the property of others.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

Intel and Intel RealSense are trademarks of Intel Corporation in the U.S. and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Copyright© 2014, Intel Corporation. All rights reserved.



04 Introduction

- Welcome
- Intel RealSense SDK Architecture
- Camera Specs
- Capture Volumes



10 Overview

- Input Modalities
- High-Level Design Principles



14 Hands

- Contour Mode
- Skeleton Tracking
- Gesture Recognition
 - Gestures
 - Common Actions
 - Supported Hand Positions
- Best Practices
 - Designing Gesture Interactions
 - How to Minimize Fatigue
- Visual Feedback
 - General Principles
 - User/World Interactions
 - Action/Object Interactions
 - Cursor Interactions



35 Face

- Face Detection
- Head Orientation
- Landmark Tracking
- Facial Expressions
- Emotion Recognition
- Avatar Control
- Use Cases
- Best Practices



44 Speech

- Speech Recognition
 - Command Mode
 - Dictation Mode
- Speech Synthesis
- Best Practices



50 Background Removal

- Background Removal
- Use Cases
- Best Practices



54 Object Tracking

- Object Tracking



57 Samples

Contents

Intel® RealSense™ technology will change how you interact- not simply with your devices, but with the world around you. You'll work and play like never before, because your devices can see, hear, and feel you.

Introduction

Welcome!

Imagine new ways of navigating the world with more senses and sensors integrated into the computing platforms of the future. Give your users a new, natural, engaging way to experience your applications. At Intel we are excited to provide the tools as the foundation for this journey with the Intel® RealSense™ SDK—and look forward to seeing what you come up with.

Over the new few months, you will be able to incorporate new capabilities into your applications including close-range hand tracking, speech recognition, face tracking, background segmentation, and object tracking, to fundamentally change how people interact with their devices and the world around them.

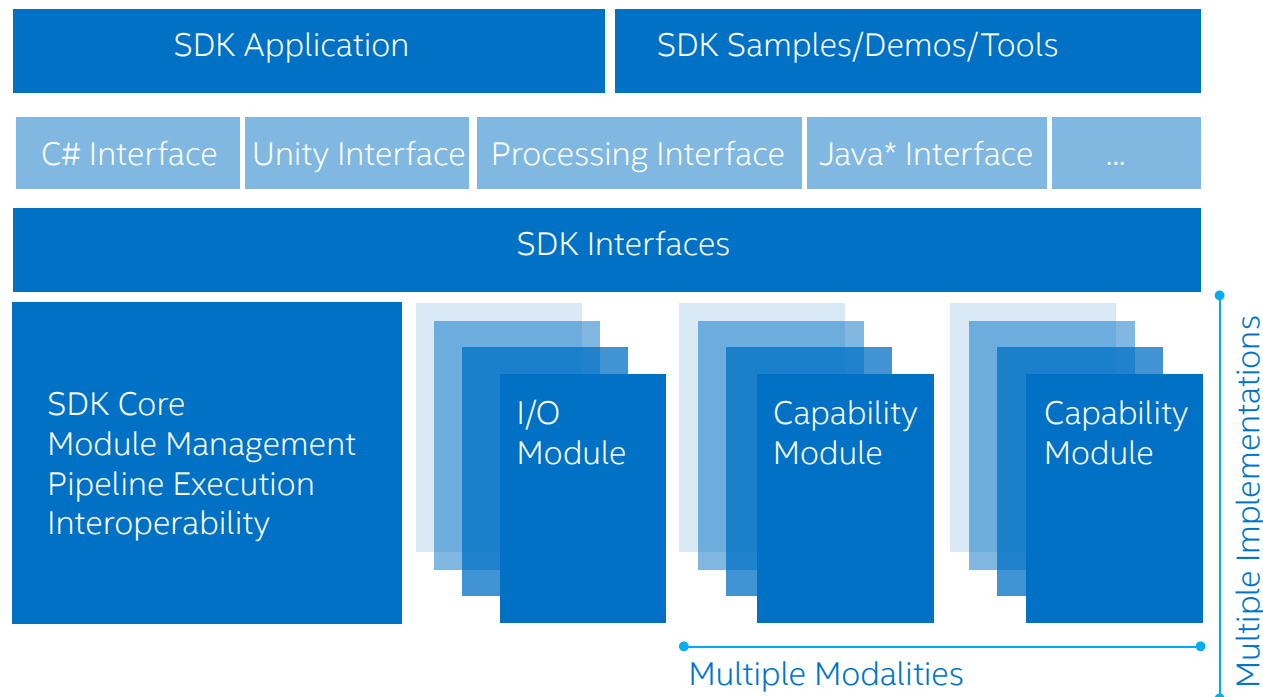


Intel RealSense SDK Architecture

Applications that integrate the Intel RealSense SDK sit on a few layers of SDK components. The base of the components is the SDK core. One of its jobs is to manage the two types of modules: input/output modules and capability modules, representing different input modalities. These modules provide the SDK functionalities to your application.

The I/O modules capture input data from your device and send that data to an output device or to the capability modules. The capability modules include various pattern detection and recognition algorithms, like face tracking and recognition, hand tracking, gesture recognition, and voice recognition and synthesis.

Another job the SDK core performs is organizing the execution pipeline. It is possible to have multiple modules contained within the pipeline at the same time, so it is essential that the pipeline have a manager. If you want to use more than one camera or other input device in your application, you may require multiple pipelines, each with its own manager.



Hardware and Software Requirements and Supported Tools

Required Hardware	A system with a minimum of a 4th generation Intel® Core™ processor, either IA-32 or Intel® 64, with integrated or peripheral depth camera
Required OS	Microsoft Windows* 8.1 OS (64-bit) Desktop Mode Microsoft Windows 8.1 New UI (Metro) Mode (coming soon)**
Supported Programming Languages	C++, C#, Java*, JavaScript
Supported IDE	Microsoft Visual Studio* C++ 2010-2013 with service pack 1 or newer
Supported Development Optional Tools	Microsoft .NET* 4.0 Framework for C# development Unity* PRO 4.1.0 or later for Unity game development Processing* 2.1.2 or later for Processing framework development Java JDK 1.7.0_11 or higher for Java development

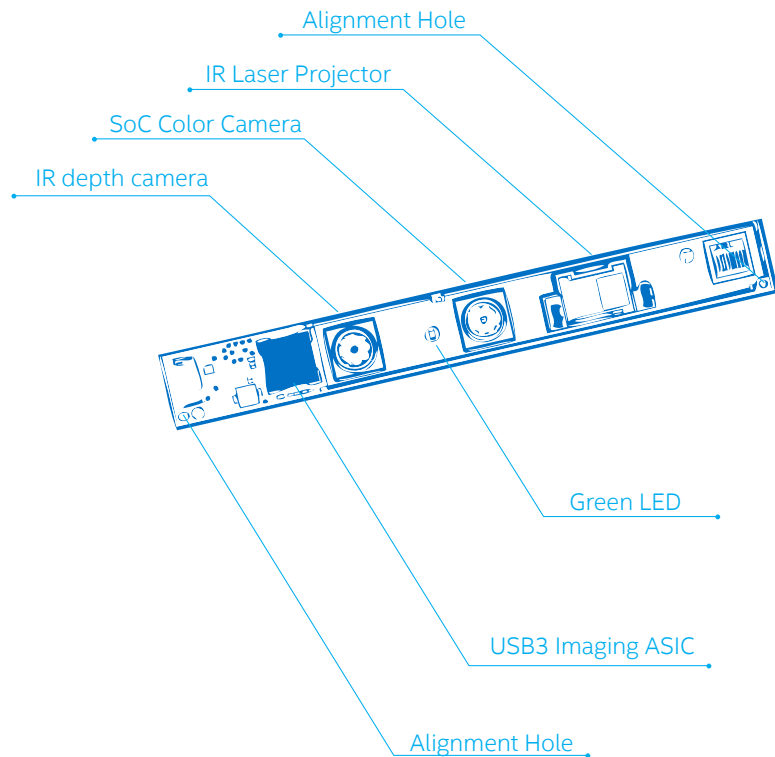
Additional SDK Features

Input Device Management	Easily share camera between applications
Multi-mode Support	Support multiple usage modes within single app (e.g., finger tracking + speech + face tracking) or between apps
Power Management State	Manage battery life
Extensible Framework	Plug in your own algorithms, add in new usage modes, and support new devices
Privacy Notification Tool	Notify user when camera is turned on by an app

* Other names and brands may be claimed as the property of others.

** Roadmap notice: All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Camera Specs



Imaging Component Overview

*Frame rate dependent on resolution
 ** Range and accuracy may vary based on targeted software usage

Color Camera	Depth (IR) Camera
Resolution	
Up to 1080p@30FPS (FHD)	Up to 640x480@60FPS (VGA), HVGA@120FPS. Up to 640x480@120FPS (IR)
Active Pixels	
1920x1080 (2M)	640 x 480 (VGA)
Aspect Ratio	
16:9	4:3
Frame Rate	
30/60/120FPS*	30/60/120FPS* (Depth), 120FPS (IR)
Field of View (DxVxH)	
77° x 43° x 70° (Cone)	90° x 59° x 73° (Cone) IR Projector FOV- N/A x 56° x 72° (Pyramid)
Color Formats	
YUV4:2:2 (Skype/Lync Modes**)	N/A
Depth Data Types	
N/A	Depth, Texture Mapping, Infrared, World Mapping

Effective Range	0.2m-1.2m
Environment	Indoor/Outdoor (Depending on Conditions)
Effective Range for Gestures	HVGA mode: 20–55cm VGA mode: 20–60cm
Effective Range for Face Tracking	2D Facial tracking: 35-120cm 3D Facial tracking: 35-70cm

Capture Volumes

The capture volume of a depth-sensing camera is visualized as a frustum defined by near and far planes and a field of view (FOV). Capture volume constraints limit the practical range of motion of the user and the physical space within which users can interact successfully. You must make sure to be aware of, and make your users aware of, the available interaction zone. Enthusiastic users can inadvertently move outside of the capture volume, so the feedback you provide must take these situations into account.



The user is performing a hand gesture inside the camera's capture volume.



The user is performing a hand gesture outside of the capture volume. The camera will not see this gesture.

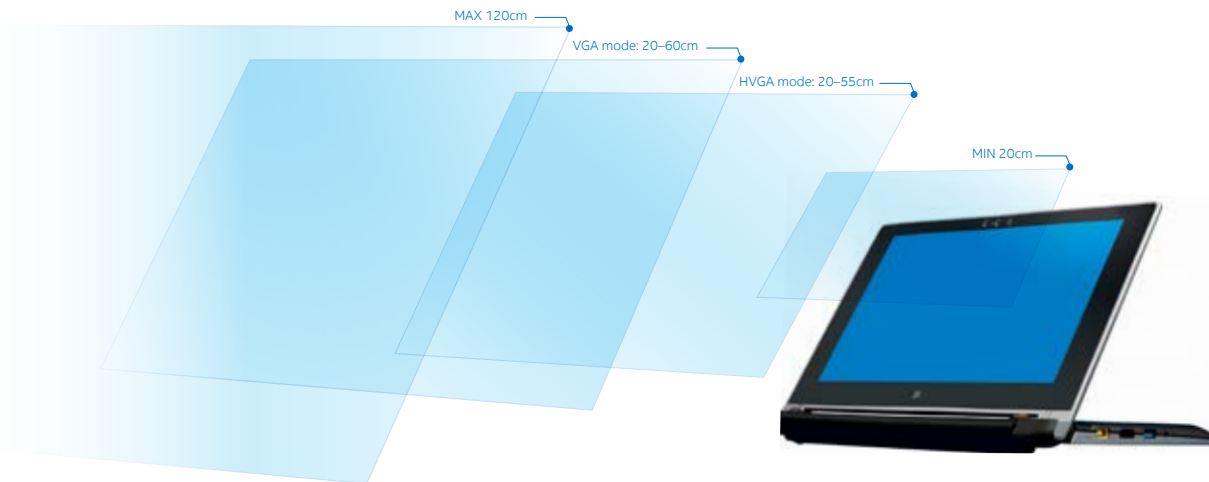
Integrated Depth Camera

There are 2 different effective ranges for skeleton tracking that your user can interact within: either short range 20-55cm (HVGA, 120FPS), or long range 20-60cm (VGA, 60FPS). You can choose which range to operate in, and can switch between them within or between apps as appropriate. Long range sees more of the hands and other objects but it runs at a lower frame rate and won't capture fine-scale or fast motions. Use short range to capture those. Fast interactions will be more common in action and first person shooter games, while most navigational interactions can be slower. .

Each of the effective ranges cater to specific interactions:

Short Range: 20-55cm (HVGA, 120FPS)	Long Range: 20-60cm (VGA, 60FPS)
For slow and fast hand movements Hand skeleton motion up to 2m/s	For slower hand movements Hand skeleton motions up to .75m/s
Single hand	2 hands

Contour mode blob tracking works up to 1m/s for up to 2 blobs in VGA mode from 20-85cm, and up to 2m/s in HVGA mode from 20-75cm. 3D face tracking works best in the 35-70cm range, while 2D face tracking works from 35-120cm,



Overview

Input Modalities

The new Intel® RealSense™ technology offers amazing opportunities to completely redefine how we interact with our computing devices. To design a successful app for this platform, you must understand its strengths. Make sure to take advantage of combining different input modalities. This will make it a more exciting and natural experience for the user, and can minimize fatigue of the hands, fingers, or voice.

Design in such a way that extending to different modalities and combinations of modalities is easy. Also keep in mind that some of your users may prefer certain modalities over others, or have differing abilities. Regardless of the input method used, it is always critical to study and evaluate how users actually engage with their devices and then to build the interface in support of those natural movements. Here's a quick rundown of some traditional and RealSense modalities, and what each is best used for:



Hands. Mid-air hand gestures allow for very rich and engaging interaction with 2D or 3D objects. They also allow easier, more literal direct manipulation. However, mid-air gesture can be tiring over long periods, and precision is limited.



Face. The best modality for sensing natural expression, emotion, and engagement. Precision is limited, and there is large variability of expressions across ages, cultures, and personalities.



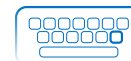
Speech. Human language is a powerful and compelling means of expression. Voice is also useful when a user is not within range of a computer's other sensors. Environmental noise and social appropriateness should be considered.



Touch. Very concrete and easy to understand, with the additional benefit of having tactile feedback to touch events and fairly good precision. However, touch is limited to 2D interaction. It is not as flexible as mid-air gesture.



Mouse. The best modality for the accurate indication of a 2D point. Large-scale screen movements can be made with small mouse movements.



Keyboard. Allows for quick and accurate text entry for adults, when voice cannot be used. Keyboard shortcuts can be quick escapes, but are not intuitive.

Intel® RealSense™-enabled Inputs

Other common inputs

High-Level Design Principles

Designing and implementing applications for the platform with Intel RealSense technology requires a very different mindset than designing for traditional platforms, such as Windows* or Mac* OS X, or even new platforms like iOS* or Android*. When designing your app, you'll want your interactions to be:

Designed to strengths. Mid-air gesture input is very different from mouse input or touch input. Each modality has its strengths and weaknesses—use each when appropriate.

Reality-inspired, but not a clone of reality. You should draw inspiration from the real world. Intel RealSense builds off of our natural skills used in every-day life. Every day we use our hands to pick up and manipulate objects and our voices to communicate. Leverage these natural human capabilities. However, do not slavishly imitate reality. In a virtual environment, we can relax the rules of the physical world to make interaction easier. For example, it is very difficult for users to precisely wrap their virtual fingers around a virtual object in order to pick it up. With the Intel RealSense SDK, it may be easier for users to perform a grasp action within a short proximity of a virtual object in order to pick it up.

Literal, not abstract. Visual cues and interaction styles built from real-world equivalents are easier to understand than abstract symbolic alternatives. Also, symbolism can vary by geography and culture, and doesn't necessarily translate. Literal design metaphors, such as switches and knobs, are culturally universal.

Intuitive. Your application should be approachable and immediately usable. Visual cues should be built in to guide the user. Voice input commands should be based around natural language usage, and your app should be flexible and tolerant in interpreting input.

Consistent. Users should perform similar operations in different parts of your application in similar ways. Where guidelines for interaction exist, as described in this document, you should follow them. Consistency across applications in the Intel RealSense ecosystem builds understanding and trust in the user.

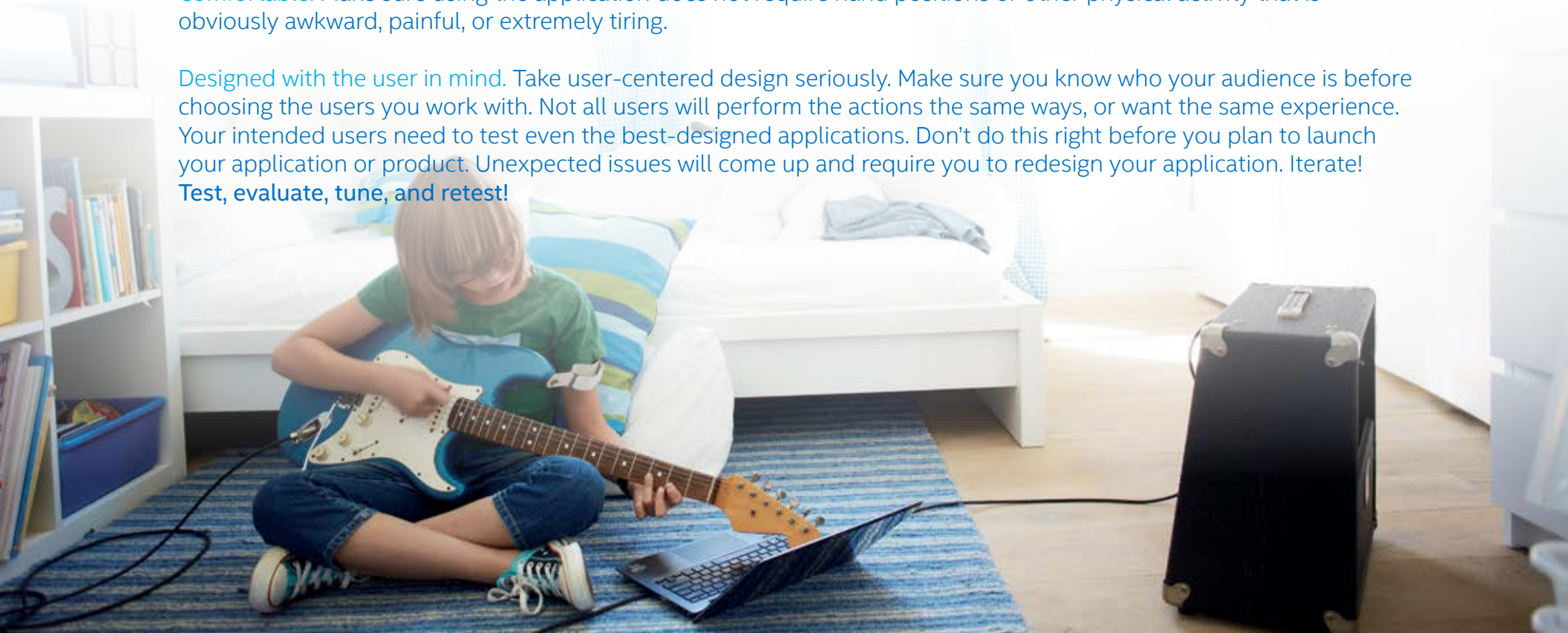
Extensible. Keep future SDK enhancements in mind. Unlike mouse interfaces, the power, robustness, and flexibility of platforms with Intel RealSense will improve over time. How will your app function in the future when sensing of hand poses improves dramatically? How about when understanding natural language improves? Design your app such that it can be improved as technology improves and new senses are integrated together.

Reliable and Recoverable. It only takes a small number of false positives to discourage a user from your application. Focus on simplicity where possible to minimize errors. Forgive your users when they make errors, and find ways to help users recover gracefully. For example, if a user's hand goes out of the field of view of the camera, make sure that your application doesn't crash or do something completely unexpected. Intelligently handle such types of situations and provide feedback.

Contextually appropriate. Are you designing a game? A medical application? A corporate content-sharing application? Make sure that the interactions you provide match the context. For example, users expect to have more fun interactions in a game, but may want more straightforward interactions in a more serious context. Pay attention to modalities (e.g., don't rely on voice in a noisy environment).

Comfortable. Make sure using the application does not require hand positions or other physical activity that is obviously awkward, painful, or extremely tiring.

Designed with the user in mind. Take user-centered design seriously. Make sure you know who your audience is before choosing the users you work with. Not all users will perform the actions the same ways, or want the same experience. Your intended users need to test even the best-designed applications. Don't do this right before you plan to launch your application or product. Unexpected issues will come up and require you to redesign your application. Iterate! **Test, evaluate, tune, and retest!**



Robust across a variety of platforms.

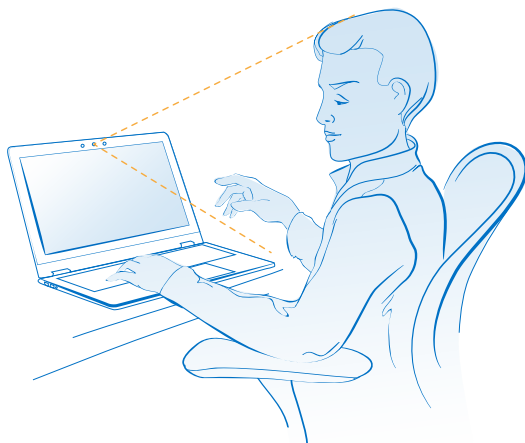
Consider the form factor your app will be running on, along with the context of your application, and design accordingly. Users might be running your application on a notebook, Ultrabook™ device, All-in-one (AIO), or 2-in-1. These different platforms present different ergonomic limitations. Keep in mind the following variables:

Screen Size

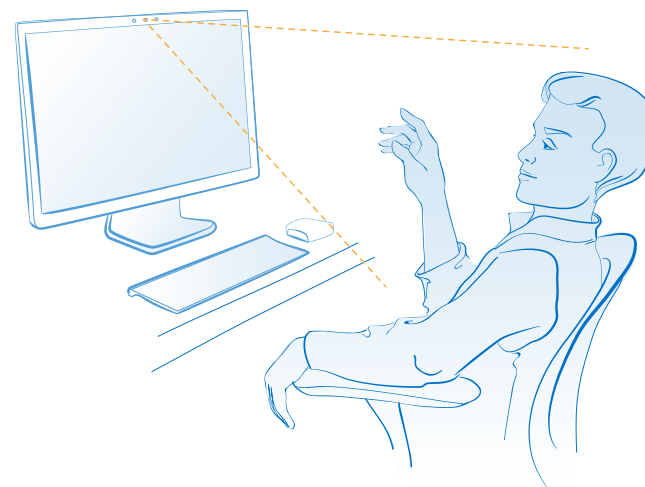
Smaller laptops and Ultrabook™ systems commonly have 13-inch screens and, occasionally, even smaller screens. AIOs may have 24-inch screens or larger. This presents a design challenge for generating application UI and related artwork and for designing interactions. You must be flexible in supporting different screen sizes.

Screen Distance

Users are normally closer to laptop screens than AIOs. Likewise, a laptop screen is often lower than an AIO screen, relative to the user's face and hands. This directly ties in to the interaction zone that you will need to design for.



When using a laptop, the user's hands tend to be very close to the screen. The screen is usually lower, relative to the user's head.



When using an AIO, user's hands are farther away from the screen. The screen is also higher, relative to the user's head.



Hands

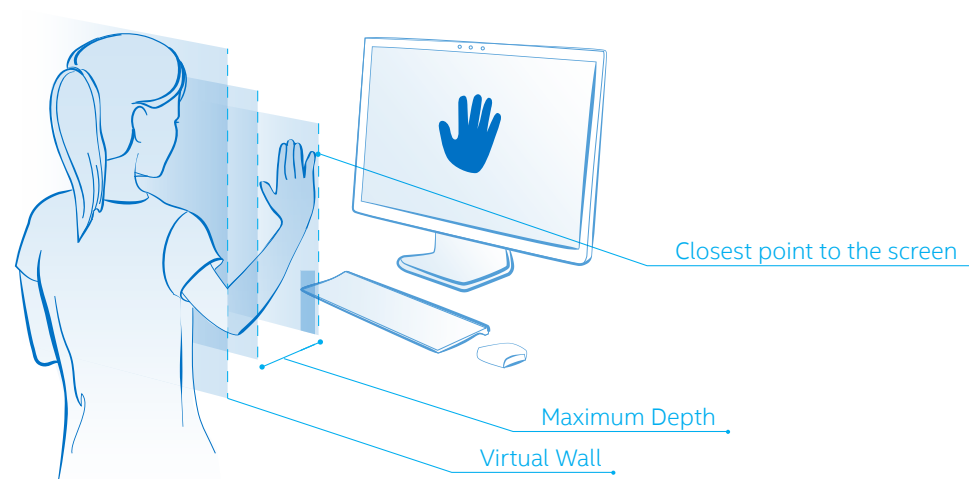
This section describes how you can use advanced 3D hand tracking with the Intel® RealSense™ SDK. It covers skeleton tracking and gestures, as well as best practices for designing and implementing mid-air gesture interactions.

- Contour Mode
- Skeleton Tracking
- Gesture Recognition
- Best Practices
- Visual Feedback

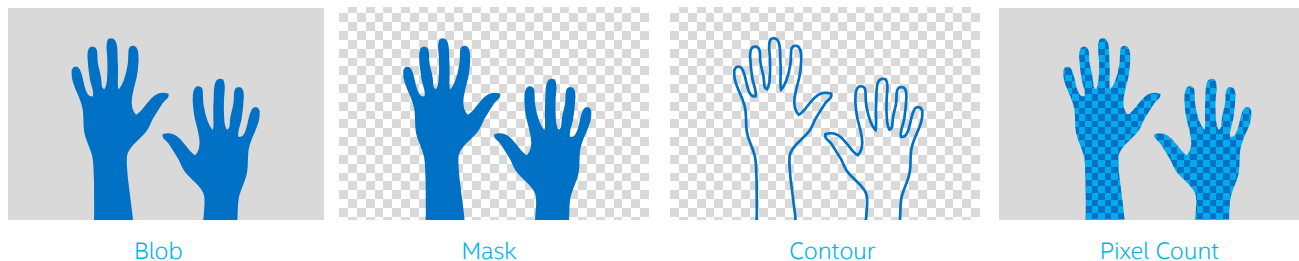
Contour Mode

The Intel RealSense SDK provides articulated hand and finger skeleton tracking, as well as object tracking. For simpler, quicker blob tracking, the SDK also has a mode called contour mode. Contour mode can track up to 2 blobs in the scene. These blobs can be anything (e.g., an open hand, a fist, a hand holding a phone, 2 hands touching) that is a connected component.

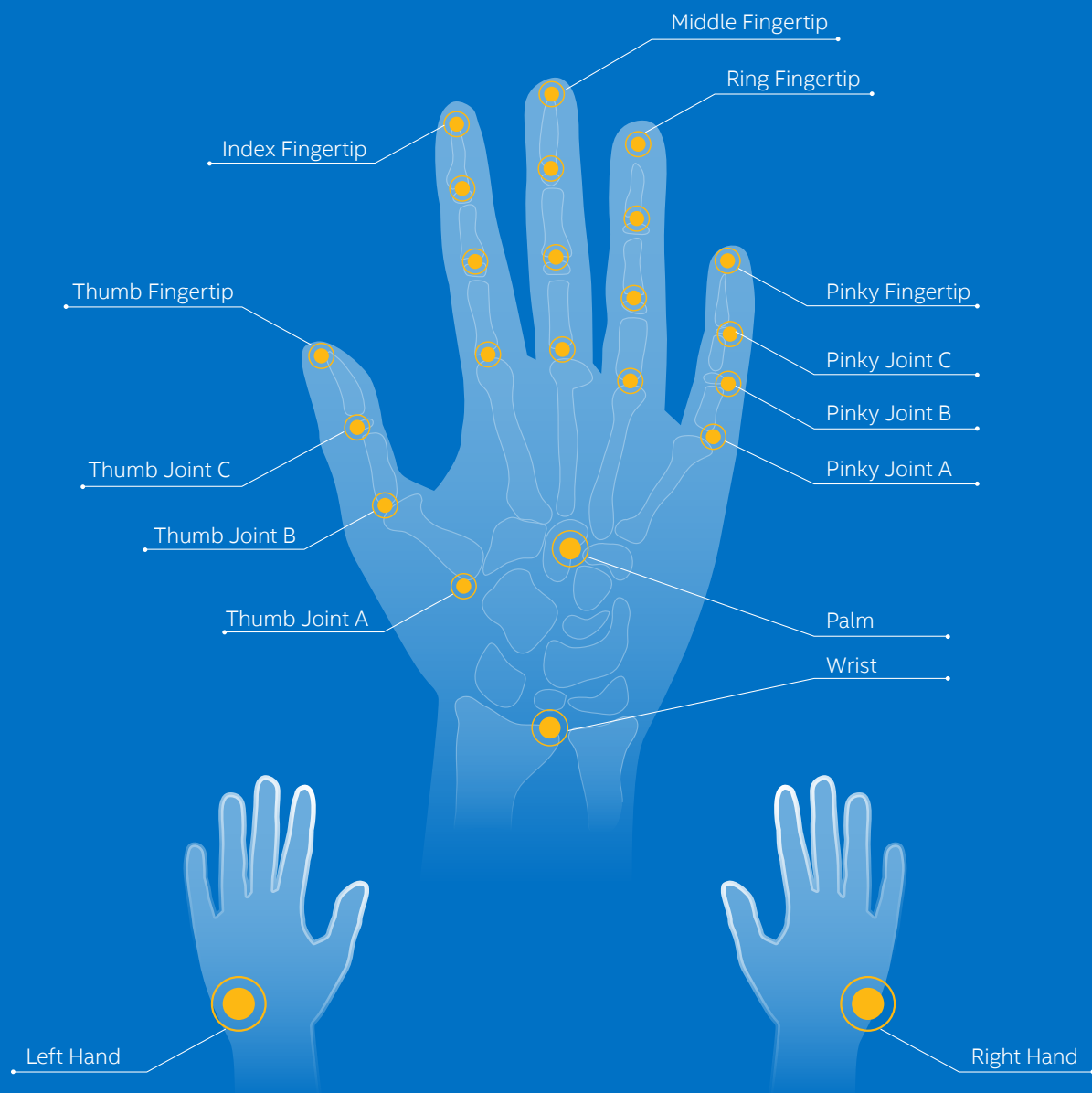
A blob is formed and selected by choosing the biggest blob that passes a virtual wall. You can adjust the distance of the virtual wall (the default is 55cm). You can also decide the max depth of the object that is to be tracked (the default is 30cm). This is useful, for example, for only segmenting the hand without the arm. You can also set the minimal blob size.



Each blob has a mask, a contour line, and a pixel count associated with it. The blob also has information about its location in 3D space and its extremity points. The blobs and contours are smoothed- this can also be controlled through the SDK.



Skeleton Tracking



As part of the Intel RealSense SDK, you have access to a precise and accurate **full-hand skeleton** which segments the hand from the background and requires no calibration. The skeleton tracks the position and orientation of **22 joints on a hand**. By having this 3D model, you can robustly handle missing information such as fingers out of the FOV and occlusions. The default skeleton is proportional to the user's hand. The SDK also provides a normalized skeleton with generic hand proportions. Different hand sizes are supported.

The **left and right hands** are labeled, as well as each finger. This allows for two-handed interactions. Up to 2 hands can be recognized in the FOV, regardless of handedness (e.g., 2 right hands can be recognized from 2 people at the same time if they are in the interaction zone). You can also access a parameter of how "folded" each of the fingers are on a spectrum of "stretched" to "folded to palm". There is also a "hand openness" parameter, a continuous value for closing a hand to a fist or full-hand pinch.

To allow continuity of the user experience and to make things more natural for the user, no calibration pose is needed for accurate hand skeleton tracking. However, it can help. If necessary, depending upon your application, you may want to include a one-time calibration step at the beginning of the app, or make it a fun, hidden part of gameplay.

Gesture Recognition

Gestures can refer to static poses (e.g., open hand), or to the movement within and between poses, commonly called gestures (e.g., wave).

The Intel RealSense SDK has a set of built-in gestures to start from. There will be a larger set of gestures as the SDK matures. All single-handed gestures can be performed with either the right or left hand. You can use these gestures as is, combine them into composite gestures, or create custom gestures that aren't included in the SDK using the hand skeleton points. If you choose to create a custom gesture, ensure that it is important to your application design. Make sure there are no conflicts in the gesture definitions that are used in the same context (e.g., if you enable Big 5 and Wave in the same level of the game, when the user waves, the spread hand gesture will be triggered as well).

Gestures

Here are the gestures currently recognized as part of the Intel RealSense SDK:



Big 5 Pose

Open the hand completely (stretch all fingers). Hand should be facing the camera and with the index/middle/ring fingers pointing upwards.



V-Sign Pose

Extend the index and middle fingers and curl the other fingers. Have some sensitivity in the hand orientation as long as its fingers are stretched up.



Tap Gesture

Keep your hand in a natural relaxed pose and move it in Z as if you are pressing a button.



Wave Gesture

Wave an open hand facing the screen. Wave length can be any number of repetitions.



Full hand pinch Pose

Pinch with all fingers extended and touching the thumb. The pinched fingers can be pointing directly to the screen or in slight profile.



Two-Finger Pinch Open Pose

Pinch with index and thumb. Hand orientation is vertical. The pinched fingers can be anywhere between pointing directly to the screen or in profile. Other fingers can be stretched or relaxed but not folded.



Fist Pose

All fingers folded into a fist. Fist can be in different orientations as long as palm is in the general direction of the camera.



Side-Swipe Gesture

Swipe is triggered when the user swipes to the left with right hand or swipes to the right with the left hand. User needs to be inside the FOV when they start to swipe. The swipe is moving the hand from one side to the other when the palm is facing the side and fingers are more or less towards the camera.



Thumbs Down Pose

Thumb stretched down and the other fingers folded. Pose can be in different orientations as long as the orientation of the thumb is down.



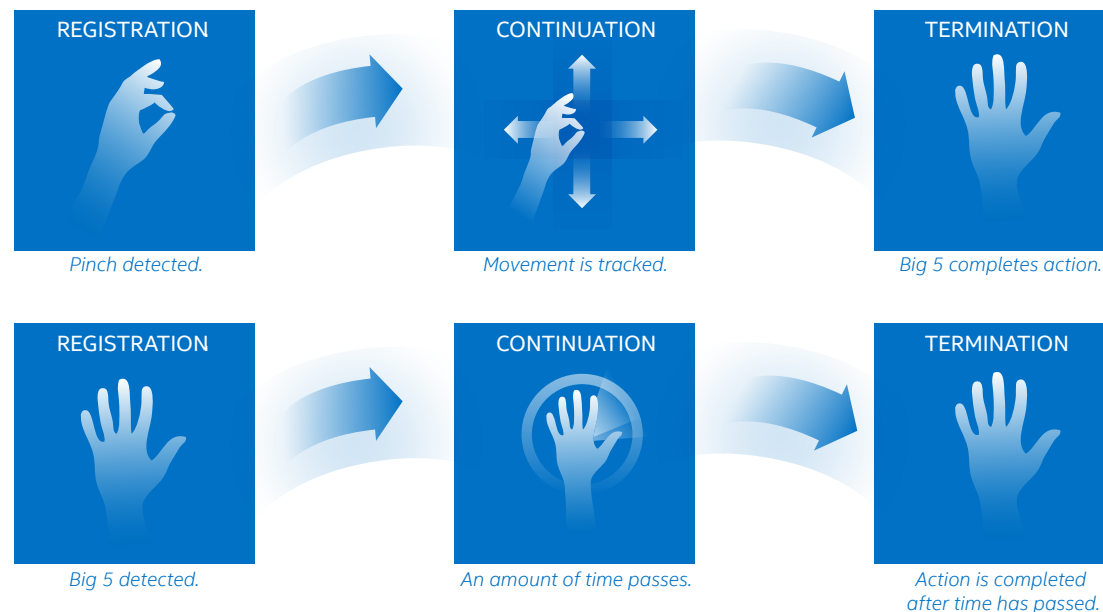
Thumbs Up Pose

Thumb stretched up and the other fingers folded. Pose can be in different orientations as long as the orientation of the thumb is up.



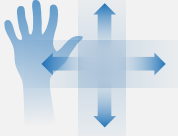




Common Actions

Think of the poses and gestures in the Intel RealSense SDK as primitives that can be used alone or in combinations to perform certain actions. Below are some common actions that may come up in many applications. When these actions exist in your application, they should generally be performed using the given gestures. Providing feedback for these gestures is critical, and is discussed in the Visual Feedback section. We don't require that you conform to these guidelines, but if you depart from these guidelines you should have a compelling user experience reason to do so. This set of universal gestures will become learned by users as standard and will become more expansive over time.

All dynamic gestures require a specific pose to register the gesture recognition system, as well as a pose to terminate the action. For actions that are only using a single SDK primitive (like Big 5, or wave), no explicit activation or closure pose is needed. There are a few ways of activating a dynamic gesture, including using a time variable, a pinch pose, an open hand pose, or having the user enter a virtual plane.



Here is a list of common actions, along with examples of how to implement them using the gestures in the SDK* or the touchless controller interface**:

Engage**		1) present your open palm to the camera in a natural pose at a distance of about 1' (30cm)
Grab and Release*		1) perform the pinch gesture to grab an object, and 2) separate the thumb and index finger to release
Scroll**		1) engage the system, 2) move the cursor with your palm to one of the screen edges, and 3) hold the cursor at the edge to scroll in that direction
Escape/Reset**		1) engage the system, and 2) wave with an open palm from side to side naturally to reset or escape from an application mode.
Go Back**		1) engage the system, and 2) rotate your hand so that the back of your hand is facing the screen
Push to Select**		1) engage the system, 2) move your open palm in parallel to the screen to get the cursor over the desired item, and 3) push forward towards the screen to select the item
Hover Select*		1) perform the Big 5 gesture, and 2) wait for x seconds to select the item

Supported hand positions

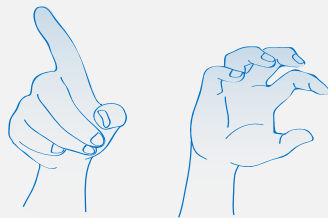
Both skeleton tracking and gesture tracking will work best in certain scenarios. Supported and unsupported single and 2-handed positions are shown below.

Single-Handed:

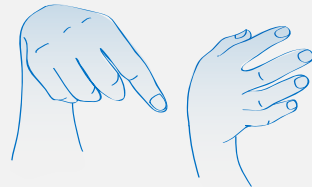
supported



Hand facing camera, fingers stretched or curled



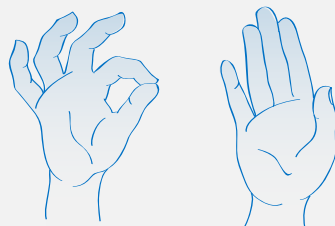
Some back hand but in general hand towards camera



Weak self-occlusions



Fingers touching, no strong self-occlusions



Strong self-occlusions



Hand holding objects



Hand touching body parts



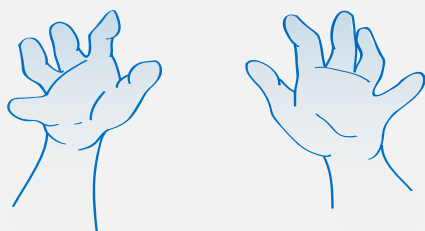
unsupported

2-Handed:

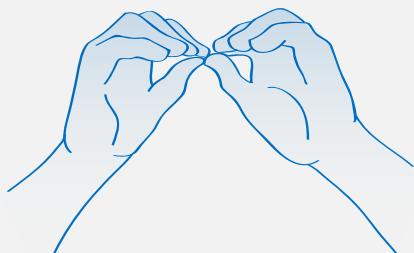
supported



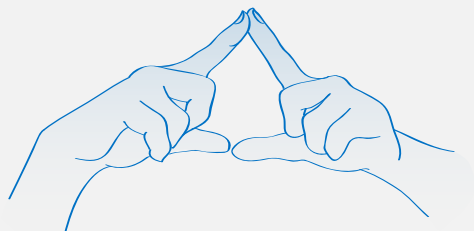
Not touching, not occluding



Close proximity, not occluding



Touching, not occluding



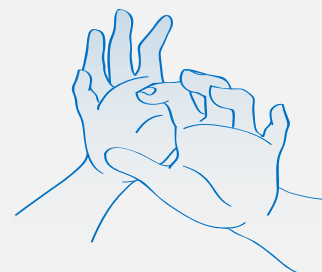
Touching, strong occlusions



unsupported



Occlusion (50% overlap), not touching



Occlusion (>50% overlap), not touching-
expect "recent position" not full tracking



limited

Other Considerations

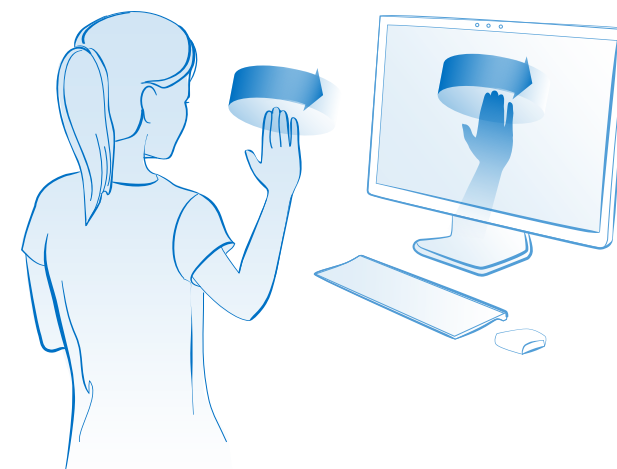
Rate- Vs. Absolute-Controlled Gesture Models

You can use an absolute-controlled model or a rate-controlled model to control gesture-adjusted parameters such as rotation, translation (of object or view), and zoom level. In an absolute model, the magnitude to which the hand is rotated or translated in the gesture is translated directly into the parameter being adjusted, e.g., rotation or translation. For example, a 90-degree rotation by the input hand results in a 90-degree rotation of the virtual object.

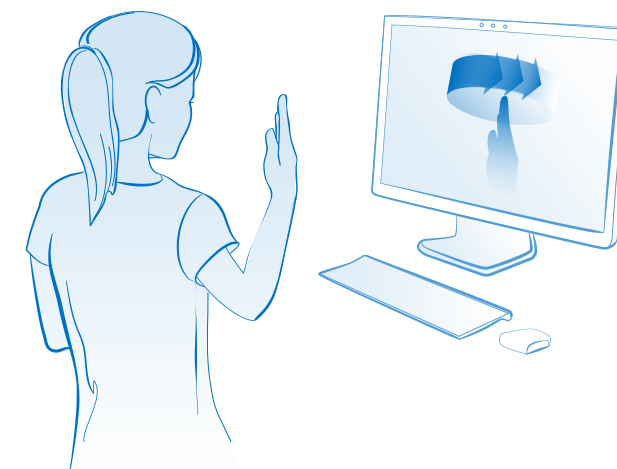
In a rate-controlled model, the magnitude of rotation/translation is translated into the rate of change of the parameter, e.g. rotational velocity or linear velocity. For example, a 90-degree rotation could be translated into a rate of change of 10 degrees/second (or some other constant rate). With a rate-controlled model, users release the object or return their hands to the starting state to stop the change.

Relative Vs. Absolute Motion

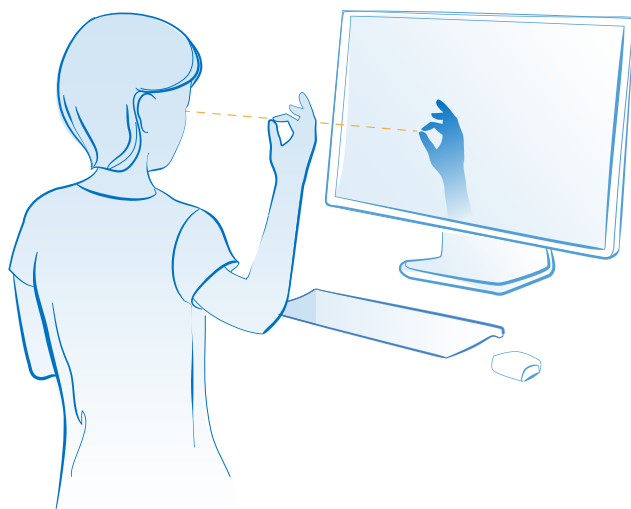
Be aware of the relationship of the interaction zone to the screen coordinates of the application. Relative motion allows the user to reset her hand representation on the screen to a location more comfortable for her hand (e.g., as one would lift a mouse and reposition it so that it is still on the mouse pad). Absolute motion preserves spatial relationships. Applications should use the motion model that makes the most sense for the particular context.



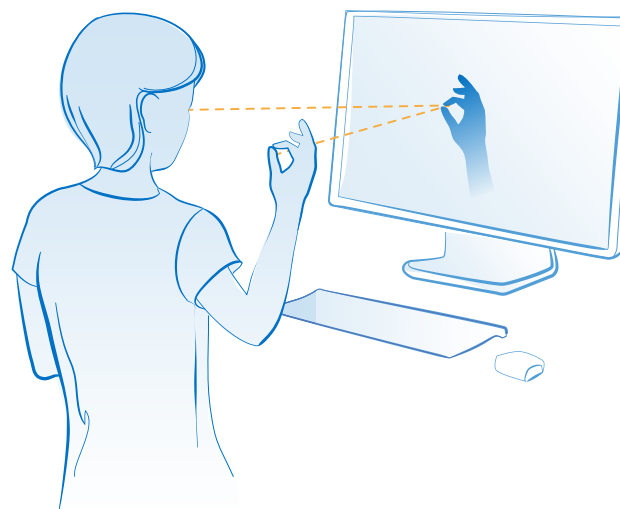
• Action on the screen directly reflects hand's position input.



• User's hand in a position that triggers an action which happens in predetermined increments.



The user's hand is mapped on the screen on the line of sight, which leads to occlusion and inability for the user to see the action on the screen.



The user's hand is mapped onto the screen away from the line of sight, and the user can naturally navigate without occluding the screen.

Occlusion

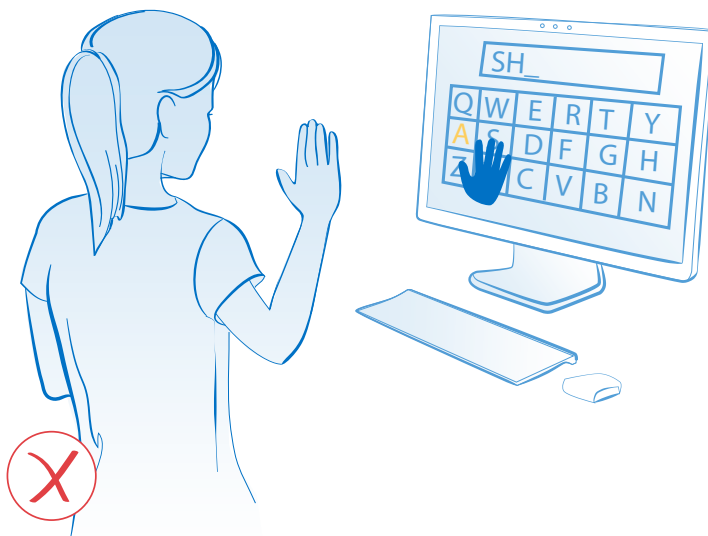
For applications involving mid-air gestures, keep in mind the problem of a user's hands occluding the user's view of the screen. It is awkward if users raise their hand to grab an object on the screen, but can't see the object because their hand caused the object to be hidden. When mapping the hand to the screen coordinates, map them in such a way that the hand is not in the line of sight of the screen object to be manipulated.



Best Practices✓

Don't model your app on existing user interfaces - build user experiences that are tuned for gesture input. Leverage the 3rd dimensionality of the hand, and don't force gesture when other methods work better.

Do not design for a mouse or touchscreen!



Typing using gesture is not convenient. In an instance when your application requires typing data, using the keyboard or touchscreen is a preferred input mode.

Moving an object in x-y-z space would be challenging to accomplish with mouse and keyboard, and is a good example of a natural experience enabled by hand tracking.

Designing Gesture Interactions



Simple

Gestures should not require unreasonable effort to learn or remember.

Where possible make use of our predefined gestures and actions. Introduce your own gesture primitives only when there is a compelling reason to do so. A small set of general-purpose natural gestures is preferable to a larger set of specialized gestures. As more apps come out, users will come to expect certain interactions, which will improve the perceived intuitiveness.

Give an escape plan. Make it easy for the user to back out of a gesture. Consider providing the equivalent of a “home button”.

Teach the gestures to the user. Some options:

- Tutorial at the beginning of the app
- Explicit in-app instructions/reminders
- Well designed visual feedback within app
- Easy help option/menu



Flexible

Support both right- and left-handed gestures.

Accommodate hands of varying sizes and amounts of motor control. Some people may have issues with the standard settings and the application will need to work with them. For example, to accommodate an older person with hand jitter, the jitter threshold should be customizable. Another example is accommodating a young child or an excitable person who makes much larger gestures than you might expect.



Comfortable

Design gestures to be ergonomically comfortable. If the user gets tired or uncomfortable, they will likely stop using your application. See next section for specific tips on how to avoid user fatigue.



Natural and Intuitive

Gestures are understandable. Use appropriate mappings of motions to conceptual actions.

As part of your app design, you need to teach the user that they are being recognized and will be interacting in a new way with the system, especially before this modality becomes commonplace.

Stay away from abstract gestures that require users to memorize a sequence of poses. Abstract gestures are gestures that do not have a real-life equivalent and don't fit any existing mental models. An example of a confusing pose is “v-sign” to delete something, as opposed to placing or throwing an item into a trash can.

Some gestures will be innate to the user (e.g., grabbing an object on the screen), while some will have to be learned (e.g., waving to escape a mode). Make sure you keep the number of learned gestures small for a low cognitive load on the user.

Design for the right space. Be aware of designing for a larger world space (e.g., with larger gestures, more arm movement) versus a smaller more constrained space (e.g., manipulating a single object).

Distinguish between environmental and object interaction.

Gestures can have different meanings in different cultures so be conscious of context when designing your app. For example, the “thumbs up” and “peace” signs both have positive connotations in North America but quite the opposite in Greece and areas of the Middle East, respectively. It may be a good idea to do a quick check on gestures before going live in other countries.

Be aware of which gestures should be actionable. What will you do if the user fixes her hair, drinks some coffee, or turns to talk to a friend? The user should not have to keep their hands out of view of the camera in order to avoid accidental gestures. Normal resting hand poses or activity should not trigger gestures.

How to Minimize Fatigue

Gestural input is naturally fatiguing as it relies on several large muscles to sustain the whole arm in the air. It is a serious problem and should not be disregarded; otherwise, users may quickly abandon the application. With these new modalities, the goal is to move away from unnatural postures and not have to conform to “technology”- to free us from awkward poses and become more natural in our interactions with technology. By carefully balancing the following guidelines, you can alleviate the issue of fatigue as much as possible:

Break up activities into small, short actions. Try to keep actions relatively brief with rest periods in between rather than having users move all over the screen. Long-lasting gestures, especially ones where the arms must be held in a static pose, quickly induce fatigue in the user’s arm and shoulder (e.g., holding the arm up for several seconds to make a selection).

Design for breaks. Users naturally, and often subconsciously, take quick breaks. Short, frequent breaks are better than long, infrequent ones.

Allow users to interact with elbows rested on a surface. Perhaps the best way to alleviate arm fatigue is by resting elbows on a chair’s armrest. Support this kind of input when possible. This, however, reduces the usable range of motion of the hand to an arc in the left and right direction. Evaluate whether interaction can be designed around this type of motion.

Do not require many repeating gestures. If you require users to constantly move their hands in a certain way for a long period of time (e.g., moving through a long list of items by panning right), they will become tired and frustrated very quickly.

Avoid actions that require your users to lift their hand above the height of their shoulder. This can get pretty tiring pretty quickly, and it can even be challenging for some users to have to lift their arms high.

Design for left-right or arced gesture movements. Whenever presented with a choice, design for movement in the left-right directions versus up-down for ease and ergonomic considerations.

Allow for relative motion instead of absolute motion wherever it makes sense. Relative motion allows the user to reset her hand representation on the screen to a location more comfortable for her hand. Do not feel the need to map the interaction zone 1:1 with the screen coordinates as this could get very tiring.

Do not require precise input. Precision is a good thing. . . up to a point. Imagine using your finger instead of a mouse to select a specific “cell” in Excel. This would be incredibly frustrating and tiring. Users naturally tense up their muscles when trying to perform very precise actions. This, in turn, accelerates fatigue. Allow for gross gestures and make your interactive objects large (see more in the Visual Feedback section).



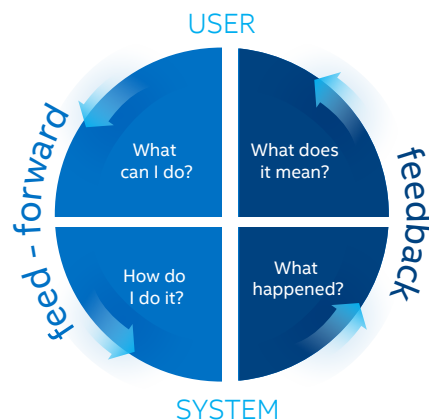
Visual Feedback

For apps that use gesture, effective feedback is especially critical. This is a novel input modality for most people, and there are no explicit physical confirmations of interaction (as with keyboard, mice, and touchscreens). When developing your app, you need to ensure that user understands how to control an application and feels the system is responsive, accurate, and satisfying.

General Principles for Good Visual Feedback

Here are some basic principles for usable feedback and effective visual communication using gestures:

Create visual UIs that consider human body ergonomics. Designers will have to step outside the existing WIMP paradigm, and create interfaces that are well attuned to human motions. One example is creating an arc-based menu that enables a user to rest their elbow on their desk while still controlling the menu.



Don Norman's user action framework

Keep in mind the cycle of interaction

Be responsive

Show feedback within 100 ms.

Be informative

Show not just what happened but what to do about it
Don't be distracting or intrusive

Use animation with natural physics

Map on-screen motion appropriately to the user's motion. For example, an upward motion of the hand should map to an upward motion on the screen, not a downward or sideways motion.

Make sure that your visual designs for icons or text feedback are legible and written to communicate effectively.



We will cover 3 general levels of interaction and feedback that you should pay attention to when building your app: User/World, Actions/Objects, and Cursors.

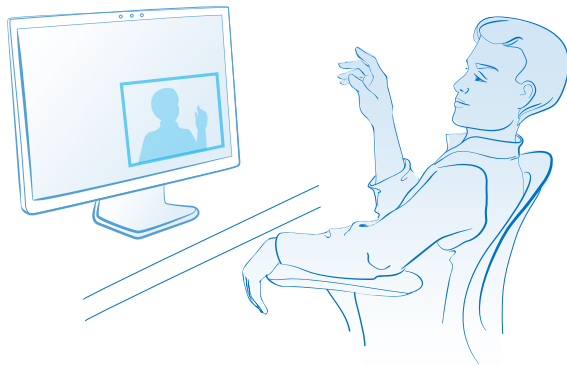
User/World Interactions

When designing your app, you need to communicate information to the user about how the camera is seeing them. Are they in the interaction zone? How should they position themselves in front of the camera?

The field of view/interaction zone of the camera is discussed in the Introduction. Here are a few recommended ways of giving user feedback- choose the one that makes sense for your particular app, and don't combine them.

View of User

User Viewport- in some apps, it can be helpful for the user to see a small viewport that shows them as the camera sees them. Recommended for: Applications that use precise hand and finger gestures. The user viewport is the best way to show users the full picture of what the system is doing with their movement, but it also can be intimidating, so don't overuse it – consider it when simpler prompts (see below) aren't enough, or as part of your application's tutorial or help experience only.



• Viewport reflects a depth image of what the camera is detecting.

User Overlay- in certain apps, a video feed of the user is helpful and appropriate. It places the user behind the 2D scene. This is recommended for games and creativity apps with the user interacting 1:1 with 2D items on the screen, or for video conferencing or editing. Design screen elements to be seen against video of the user. Mirror the video to match the user's motions.



• User is shown with a 2D color camera feed and has the ability to interact with 2D objects or interface items.

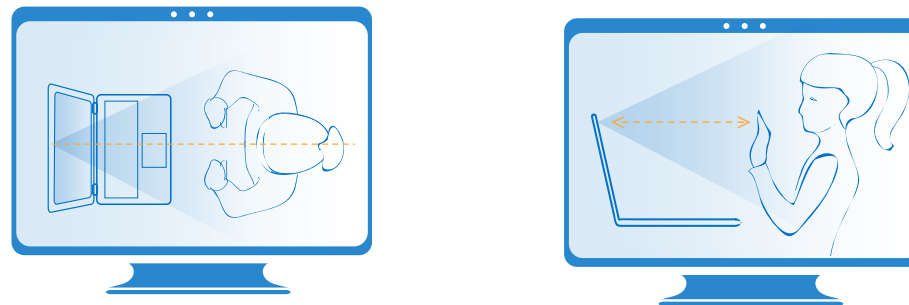
Distance Prompts

Simple prompts indicate the near and far boundaries of the interaction zone. Without prompts, users see the system become unresponsive and don't understand what to do next. Filter the distance data and show the prompt after a slight delay. Use positive instructions instead of error alerts.



World Diagrams

World diagrams orient the user and introduce them to the notion of a depth camera with an interaction zone. This is recommended for help screens and tutorials, and games for users new to the camera. Don't show this every time, only during a tutorial or on a help screen. Don't make it too technical and consider the audience.



Introduce the user to the optimum interaction range, using diagrams similar to the above.

Action/Object Interactions

When designing apps for gesture, you need to consider how to visually show the user how they will be interacting with objects on the screen.

Actionable Buttons

Show what to do and what is actionable. Use color and animation to draw objects according to interaction state (e.g., hover, selected, not selected). Distinguish actionable 3D objects from the rest of the scene. Use standard selection gestures, and suggest the select action.

Press



Navigating towards an actionable button



Pushing hand towards the screen to select

Hover



Navigating towards an actionable button



Holding hand still for a specified amount of time



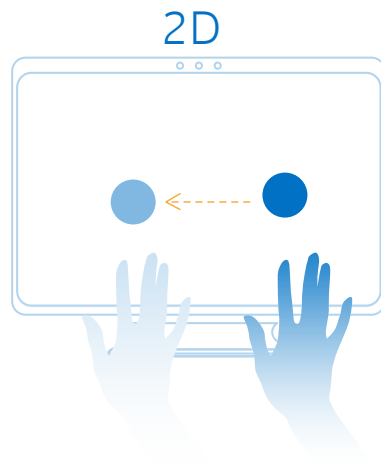
Use indicator to show the progress.

Grabbable Objects

- Objects should be large enough to account for slight hand jitter
- Objects should be far enough apart so users won't inadvertently grab the wrong object.
- Avoid placing interaction elements too close to the edge of the screen, so the user doesn't get frustrated with popping out of the field of view.
- If the interface relies heavily on grabbing and moving, it should be obvious to the user where a grabbed object can be dropped
- If the hand becomes untracked while the user is moving an object the moved object should reset to its origin, and the tracking failure should be communicated to the user.

Consider the dimensions of input

2D input is most robust and will be most familiar to users. Since we are currently using 2D displays, use it for all simple interactions, especially in productivity apps. 2.5D adds simple depth input for special actions. Full 3D-space interaction is most challenging for users. To show that the user is operating in a 3D space, aggressively use depth cues- shadowing, shading, texture, and perspective.



Reading inputs in x-y axis



Detecting inputs in x-y axes and z input for simple actions



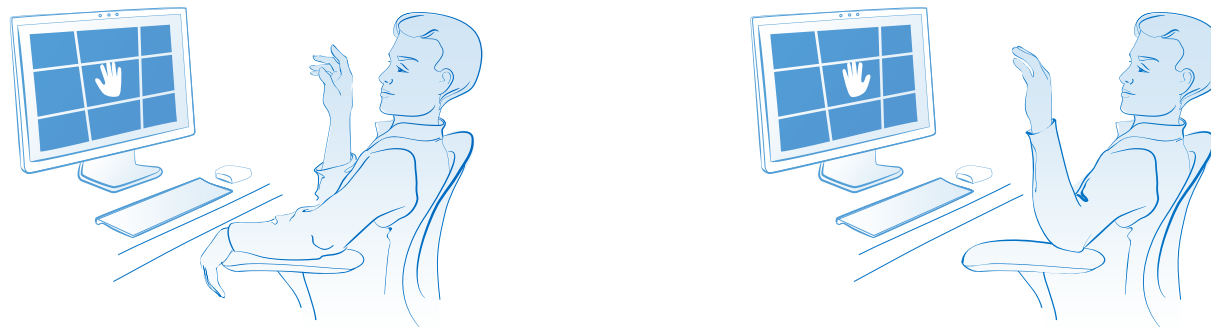
Full 3D input tracking

Cursor Interactions

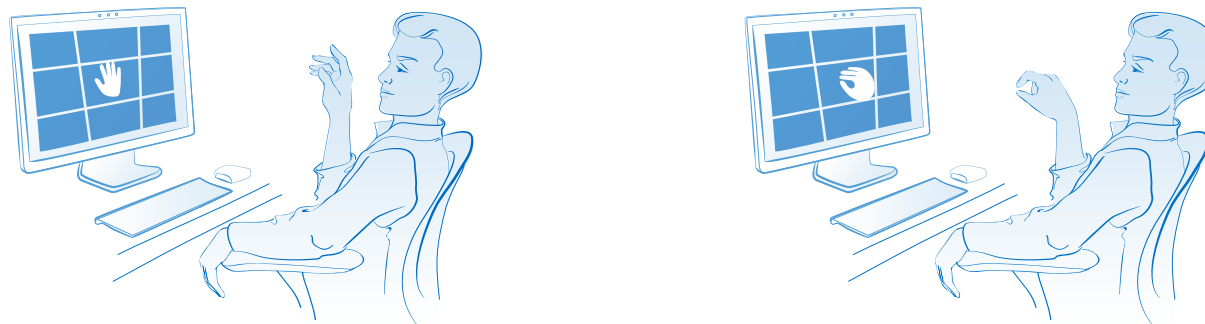
For air gestures, users prefer a cursor that mimics their hand over an abstract design. Cursors are recommended for apps that do pointing and selecting. Do not use a cursor for apps that implement a user overlay or user viewport, or with apps that use heavy 3D interaction. A 2D visual cursor assumes that the user is mostly operating on a plane in 3D space that is parallel to the screen. Make sure to smooth the cursor motion to avoid visual jitter. Animate the cursor to show the appropriate level of detail. If relevant to your interactions, you should show individual finger motions.

A cursor will focus the user's attention; don't use it if it's not necessary. Put important alerts about system state at the cursor.

Dynamic Hand Cursor



Here is an example of a dynamic hand cursor showing the left or right hand according to which is seen.



Here we see the dynamic hand cursor giving visual feedback that the user has made a pinch gesture.



Face

This section describes how you can use advanced 2D and 3D face tracking with the Intel® RealSense™ SDK. It covers face detection, head orientation, landmark tracking, avatar control, and emotion recognition, as well as best practices for designing and implementing interactions that use face data.

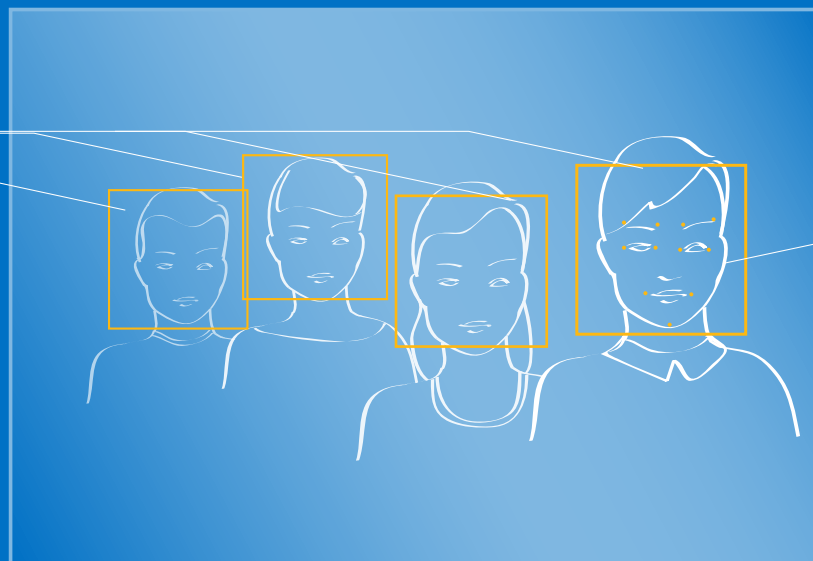
- Face Detection and Recognition
- Head Orientation
- Landmark Tracking
- Facial Expressions
- Emotion Recognition
- Avatar Control
- Use Cases + Best Practices



Face Detection

The Intel RealSense SDK provides accurate 3D detection and tracking of all faces in the scene at a range of up to 1.2 meters. There is a maximum number of 4 faces that can be tracked. You can choose which 4 to detect (e.g., 4 closest to the screen, 4 most right). Each marked face will be outlined with an approximated rectangle (you can get the x, y, and z coordinates of this rectangle). Compared to 2D tracking, 3D head tracking capability maintains tracking with much greater head movement. It works in wide lighting conditions. You can get alerts for when the face is in the FOV, partially in the FOV, or occluded.

Up to four faces
can be tracked at the same time



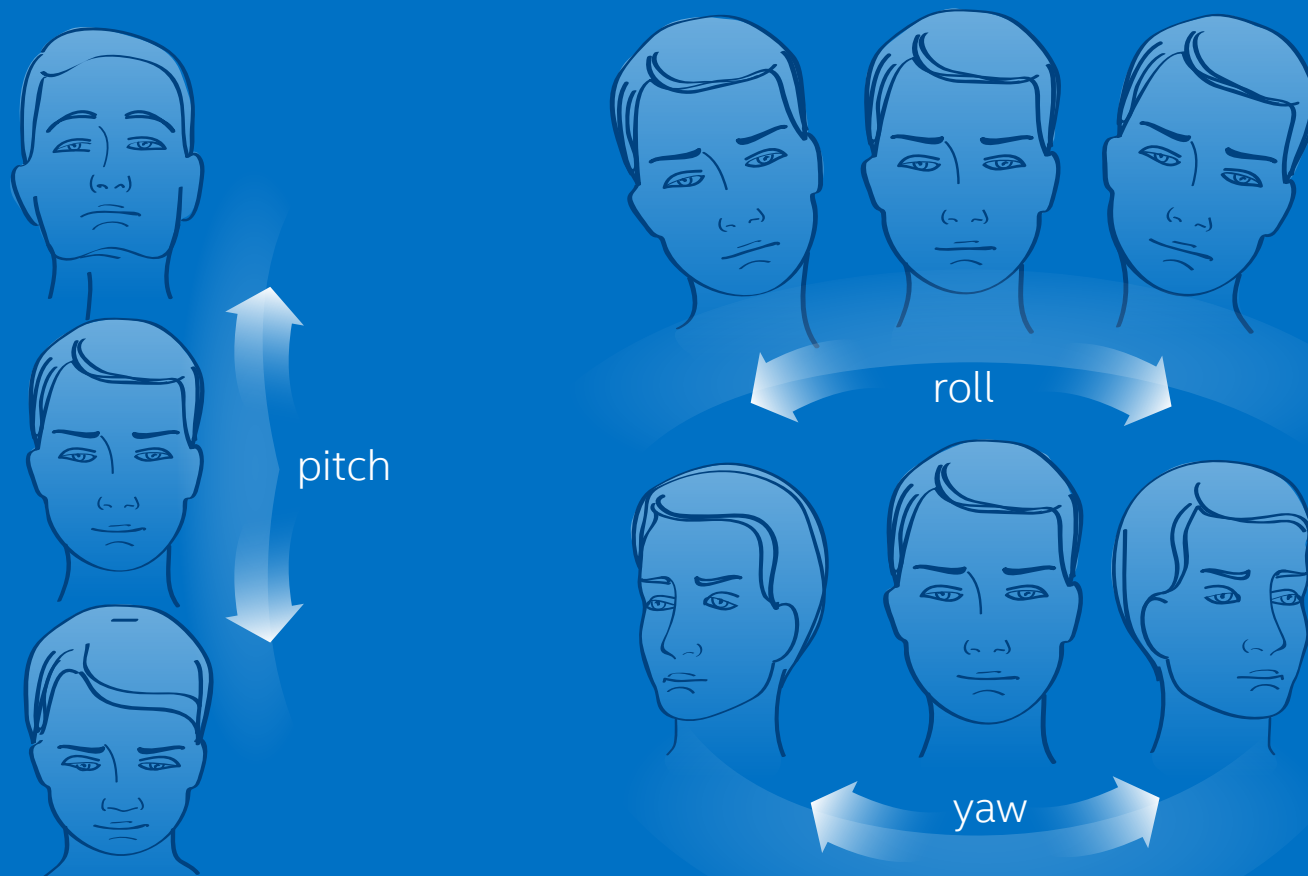
One face has
complete landmark tracking

Face Recognition

The SDK provides the ability to recognize specific faces. Once a face is registered, an ID is assigned to it and some information about it is stored in the memory of the Face library. If the same face is registered multiple times, it will improve the chances of correct recognition of that face in the future. Whenever there is an unrecognized face in the frame, the recognition module will compare it against the available data in the database, and if it finds a match it will return the stored ID for that face.

Users do not need to worry about their images being stored. The data saved is a collection of features gathered from the image by the algorithm.

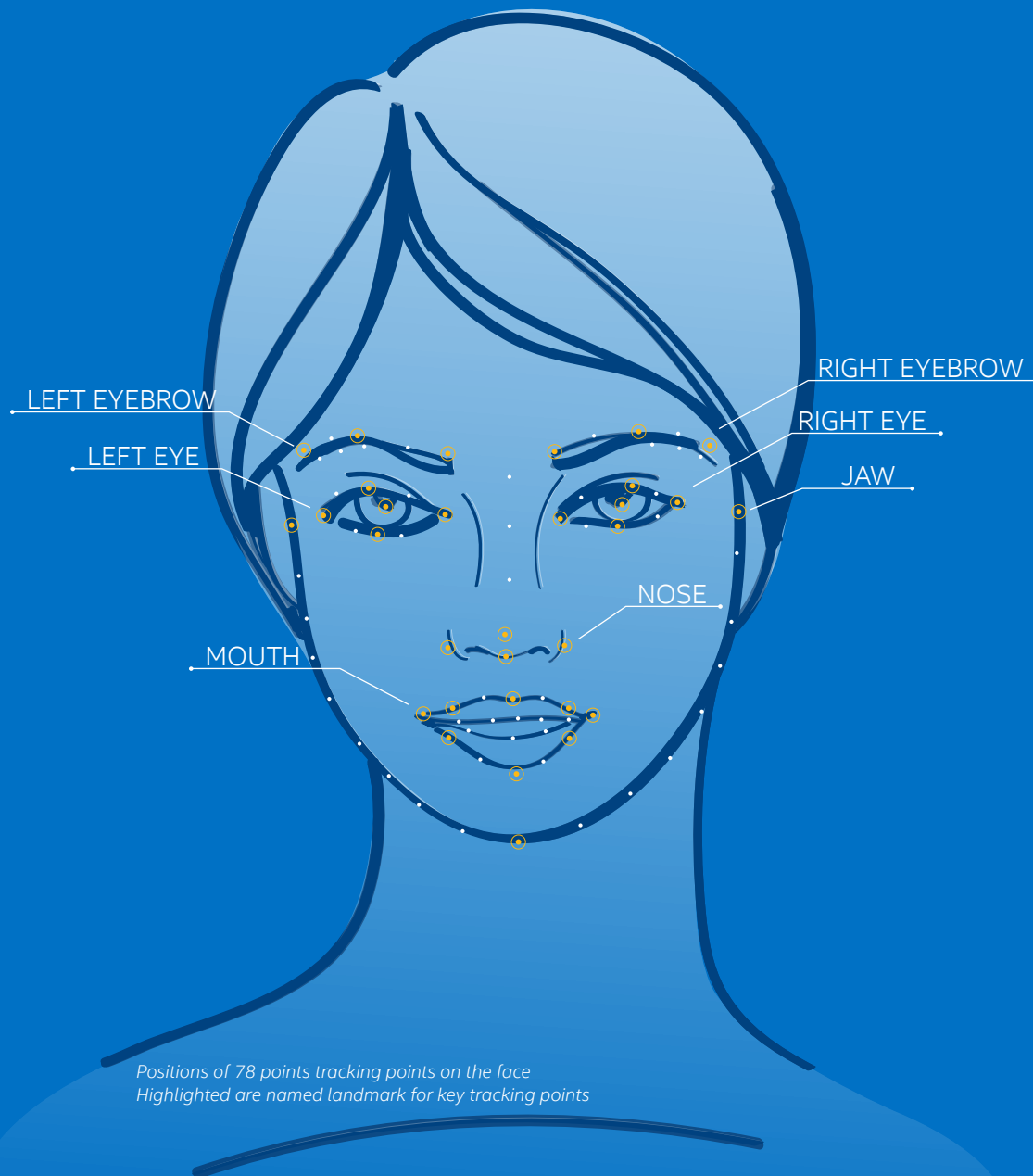
Head Orientation



The SDK provides you with the 3D orientation of the head. This gives you an idea of where the user's head is pointing (in some cases, you could infer they are likely looking in that direction as well). You can experiment with this as a very coarse version of eye tracking. Stay tuned for finer-tuned gaze tracking in the next release! For now, tracking head movements works best on the yaw and pitch axes.



Landmark Tracking



Positions of 78 points tracking points on the face
Highlighted are named landmark for key tracking points

The SDK provides 3D tracking of 78 points of interest on the face for accurate facial attribute recognition and analysis. The position of the points of interest are provided in image and world coordinates. Facial landmark tracking supports avatar creation and facial animation and mimicry, as well as simple expression recognition. You can use this points directly, or use relative positions of the points to trigger actions.

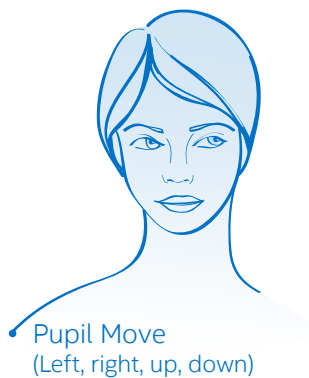
You have easy access to a subset of face landmark regions (left and right eyes, left and right eyebrows, nose, mouth, and jaw). Labeled landmarks are shown in their relative regions.

Landmark tracking is supported for users with and without facial hair and glasses. Initialization (when the face first enters the FOV and is being detected) works best with the frontal face +/- 15 degrees on the yaw axis. Roll and pitch should be close to 0 degrees. Landmark tracking works best when the user's head is within 30 degrees of the screen (both yaw and pitch).

Facial Expressions

The SDK also includes higher-level facial expression recognition. This can make the creation of cartoonish avatars easier. Each of the expressions have an intensity level from 0 to 100 to allow for smoother, more natural animation.

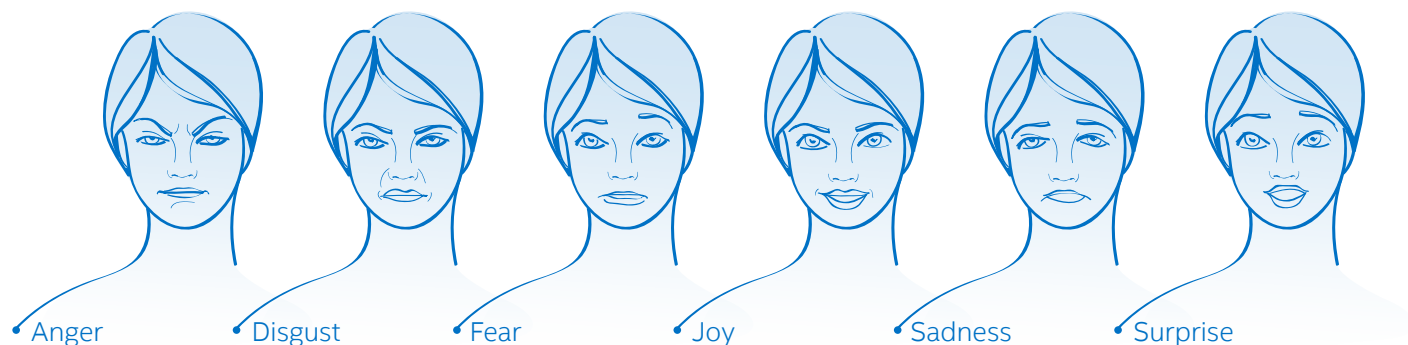
The following expressions are in the SDK:



Emotion Recognition

Emotient's emotion recognition algorithms in our SDK use 2D RGB data. The Emotion module is not part of the Face module. The faces in an image or video are located in real-time up to 30fps, and interpolation and smoothing of face data is extracted from consecutive frames of a video. For emotion recognition to work, the faces in the image have to be at least 48x48 pixels. You can query for the largest face in the FOV. The algorithms are not limited to RGB data and can be used with greyscale data as well.

With the SDK, you can detect and estimate the intensity of the six primary expressions of emotion (anger, disgust, fear, joy, sadness, surprise) seen below:



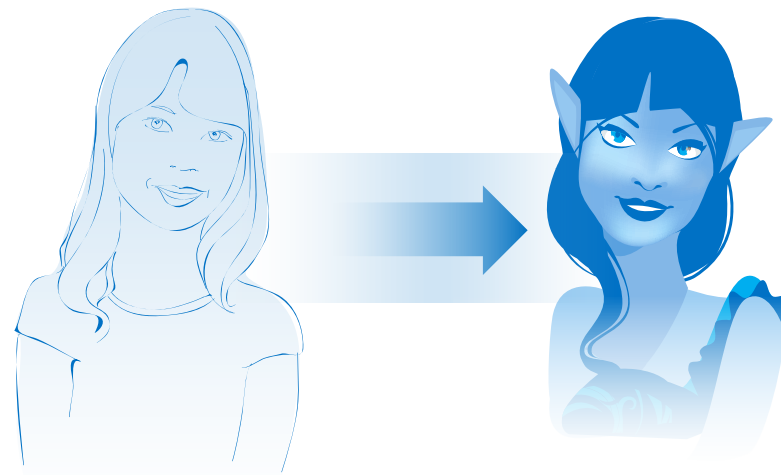
The emotion channels operate independently of one another. There are 2 ways to access the channel data- either by intensity (between 0 and 1) or evidence (the odds in log scale of a target expression being present). See the SDK documentation for more details. You can also access aggregate indicators of positive and negative emotion. There is a neutral emotion channel to help calibrate the other channels, and an experimental contempt channel. Currently, joy, surprise, and disgust are the easiest emotions to recognize.

Facial artifacts, such as glasses or facial hair, may make it advisable to focus on changes in the emotion outputs rather than absolute values. For example, a person may have facial hair that makes him appear less joyful than he would if he did not have facial hair.

You can combine a few emotions together to give a guess of negative emotion. For example, you could combine varying intensities of disgust, fear, and anger and assume your user is likely expressing a negative emotion.

Avatar Control

The SDK provides simple avatar control for use in your applications by combining the facial expressions and landmarks available in the Face module. The SDK provides sample code for a Character Animation that enables your application to use any face model and animate the user as part of your application. All the code, assets and rigging are part of the code distribution of the SDK.



You can:

- Add your own avatar models

- Adjust smoothing on the landmarks and expressions for various usages and behaviors

- Enable/disable specific expressions or landmark subsets (e.g., you may want to only move the eyes on your avatar)

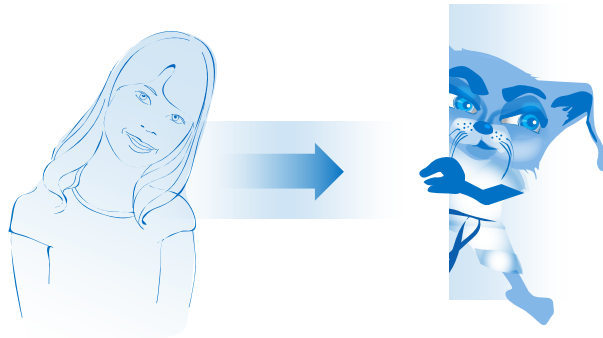
- Allow mirroring (e.g., when the user blinks their right eye, the left eye of the avatar will blink)

- Tune the resolution of the animation graphics

- Plug the avatar into different environments (you can provide your own background, e.g. an image, video, or camera stream)

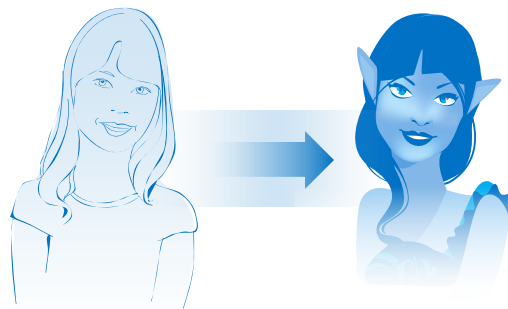
Use Cases

There are many reasons why you might want to use head tracking, face tracking, and emotion recognition in your apps. We outline a few examples that can work well leveraging the Intel RealSense SDK:



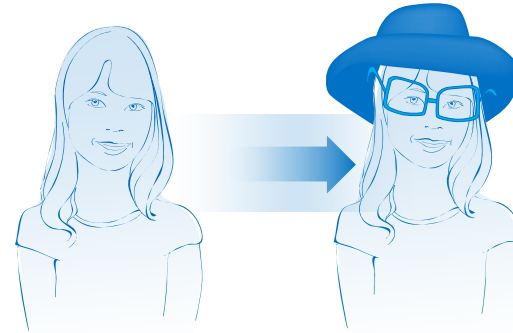
Gaming/App Enhancements

Use head tracking and orientation to allow navigation, parallax, view changes, or to peek around corners. Use landmark tracking to identify user's expressions.



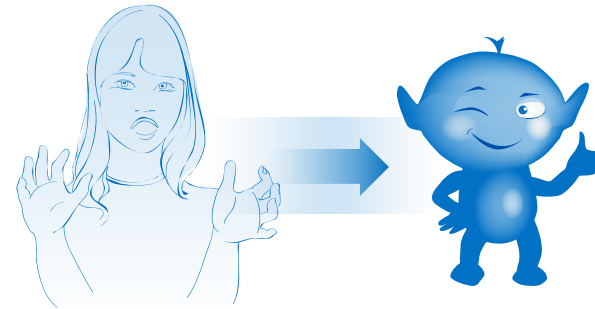
Avatar Creation

Create cartoonish or realistic looking avatars that mimic the user's face. It is recommended to stick to more abstracted or cartoonish avatars, to avoid the uncanny valley effect and to have more robust and predictable facial mimicry.



Face Augmentation

Use head tracking to augment the user's face on the screen in either a cartoonish (e.g., wear a wizard hat) or realistic (e.g., try out new kinds of glasses) way.



Affective Computing

Identify and respond to user's mood and level of engagement, either implicitly or explicitly.



Best Practices✓



Lighting and Environment

Good lighting is very important especially for 2D RGB tracking

For optimal tracking:

- use indoor, uniform, diffused illumination

- have ambient light or light facing the user's face (avoid shadows)

- avoid backlighting or other strong directional lighting



User Feedback

Give feedback to the user to make sure they are in a typical working distance away from the computer, for optimal feature detection.

Notify the user if they are moving too fast to properly track facial features.



User Interactions

Design short interactions

Avoid user fatigue. Do not ask or expect the users to move their head/neck quickly or to tilt their head on any axis more than 30 degrees from the screen. If you are asking the user to tilt their head, make sure they can still see relevant cues and information on the screen.

Test emotion detection and landmark-based expression detection by testing with your audience. People express their emotions very differently based on culture, age, and situation. Also, remember that every face may have a different baseline for any given emotion.



User Privacy

Communicate to users about where images go and what happens to them. For most applications, you will not need to save the actual images, but could save the information about them (e.g. emotion intensity, landmark coordinates, angle, size)



Speech

This section describes best practices for designing and implementing voice command and control, dictation, and text to speech for your applications. As of now, English is the only supported language, and speech recognition works best for adults.

Speech Recognition
Speech Synthesis
Best Practices

Speech Recognition



Command Mode Vs. Dictation Mode

Be aware of the different listening modes your application will be in.
Once listening, your application can be listening in command mode or dictation mode.

Command Mode

Command mode is for issuing commands tied to discrete actions (e.g., saying “fire” to trigger cannon fire in a game).

((("e-mail photo" "fire!" "open" "search")
"mute" "next" "save"))

In command mode, the SDK module recognizes only from a predefined list of context phrases that you have set. The developer can use multiple command lists, which we will call grammars. Good application design would create multiple grammars and activate the one that is relevant to the current application state (this limits what the user can do at any given point in time based on the command grammar used). You can get recognition confidence scores for command and control grammars. To invoke the command mode, provide a grammar.

When constructing your grammars, keep them:



Natural.

The language you use is very important. Ask other people – friends, family, people on forums, study participants – how they would want to interact with your system or initiate certain events.

Responsiveness is important; experiment with different pause lengths at the end of sentences.



Flexible.

Provide many different options for your grammar to exert less effort on the user. For example, instead of constraining the user to say “Pause music”, you could also accept “Pause my music”, “Pause song”, “Stop music”, etc.



Simple, yet distinct.

Complicated words and names are not easily recognized. Make your grammar include commonly used words. However, very short words can be difficult to recognize because of sound ambiguity with other words (e.g., “yes” and “guess”). Be aware of easily confusable commands. For example, “Create playlist” and “Create a list” will likely sound the same to your application. One would be used in a media player setting, and the other could be in a word processor setting, but if they are all in one grammar the application could have undesired responses.

Dictation Mode

Dictation mode is for open-ended language from the user (e.g., entering in the text for a Facebook status update). Dictation mode has a predefined vocabulary. It is a large, generic vocabulary containing 50k+ common words. Highly domain specific terms (e.g., medical terminology) may not be widely represented in the generic vocabulary file, but you can customize your app to a specialized domain if desired. You can add in your own custom vocabulary to the dictation engine if you find that specific words you need are not in the dictionary.

50k common words ⌚ 30sec limit

Absence of a provided command grammar will invoke the SDK in dictation mode. Dictation is limited to 30 seconds. Currently, grammar mode and dictation mode cannot be run at the same time.



Speech Synthesis

You can also generate speech using the built in Nuance speech synthesis that comes with our SDK. Currently a female voice is used for text-to-speech (TTS). Speech can be synthesized dynamically.

Make sure to use speech synthesis where it makes sense. If there is a narrator or a character speaking throughout your application, it may make more sense to pre-record speech where dynamic speech synthesis isn't needed. Have an alternative for people who cannot hear well, or if speakers are muted.





Best Practices✓

Be aware of speech's best uses. Some great uses of speech are for short dictations, triggers, or shortcuts.

For example, speech could be used as a shortcut for a multi-menu action (something that requires more than a first-level menu and a single mouse click). However, to scroll down a menu, it may make more sense to use the touchscreen or a gesture rather than repeatedly have the user say "Down", "Down", "Down".

Here are some best practices for using speech in your applications:



Environment

Be aware that speech can be socially awkward in public, and background noise can easily get in the way of successful voice recognition.

Test your application in noisy backgrounds and different environmental spaces to ensure robustness of sound input.



Naturalness/Comfort

People do not speak the way they write. Be aware of pauses and interjections such as "um" and "uh".

Don't design your application such that the user must speak constantly. Make verbal interactions short, and allow for breaks to alleviate fatigue.

Listening to long synthesized speech will be tiresome. Synthesize and speak only short sentences.



User Control

Watch out for false positives- some sounds could unexpectedly crop up as background noise. Do not implement voice commands for dangerous or unrecoverable actions (such as deleting a file without verification).

Give users the ability to make the system start or stop listening. You can provide a button or key press for “push to talk” control.

Give users the ability to edit/redo/change their dictation quickly. It might be easier for the user to edit with the mouse, keyboard, or touchscreen at some point to edit their dictations.



Feedback

Teach the user how to use your system as they use it. Give more help initially, then fade it away as the user gets more comfortable (or have it as a customizable option).

Always show the listening status of the system. Is your application listening? Not listening? Processing sound? Let the user know how to initiate listening mode.

Let the user know what commands are possible. It is not obvious to the user what your application's current grammar is. This information can be shown in an easily accessible

Let the user know that their commands have been understood. The user needs to know this to trust the system, and know which part is broken if something doesn't go the way they planned. One easy way to do this is to relay back a command. For example, the user could say “Program start”, and the system could respond by saying “Starting program, please wait”.

If you give verbal feedback, make sure it is necessary, important, and concise! Don't overuse verbal feedback as it could get annoying to the user.

If sound is muted, provide visual components and feedback, and have an alternative for any synthesized speech.



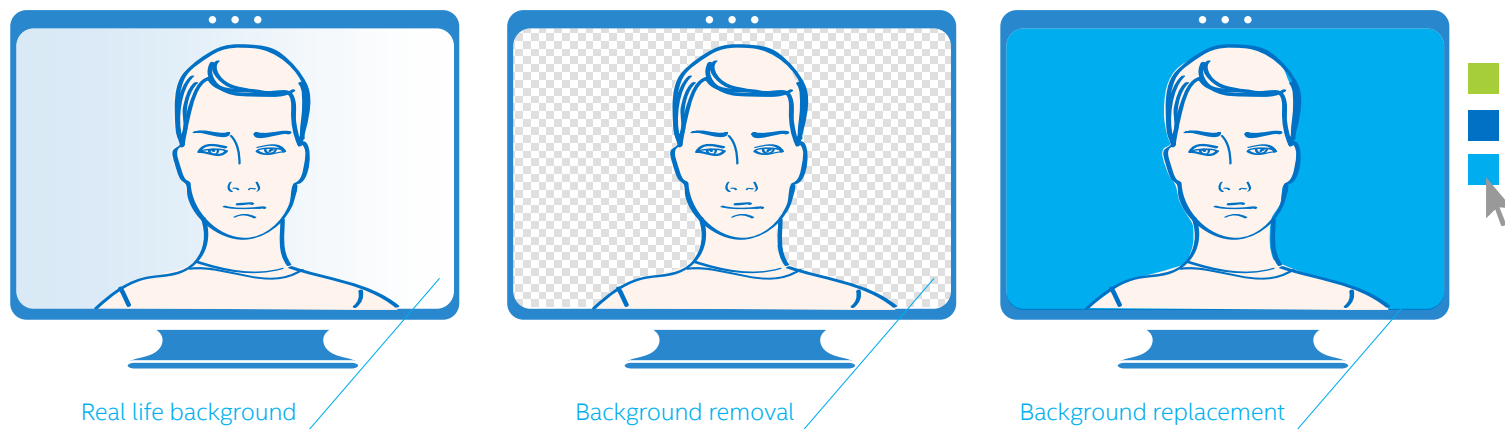
Background Removal

This section describes how you can use advanced user/background segmentation techniques with the Intel® RealSense™ SDK.

User Segmentation
Use Cases
Best Practices

Background Removal

The Intel RealSense SDK allows you to remove users' backgrounds without the need for specialized equipment. You can mimic green-screen techniques in real-time without post-processing. A segmented image can be generated per frame which can remove or replace portions of the image behind the user's head and shoulders. If the user is holding an object, this will also be segmented.

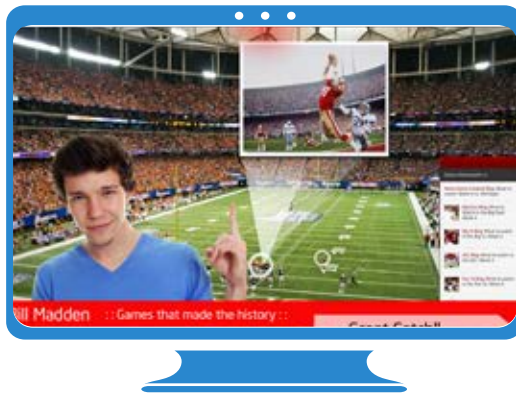


The resolution of the segmented output image matches the resolution of the input color image, and contains a copy of the input color data and an alpha channel mask. Pixels that are part of the background have an alpha value less than 128, and pixels that correspond to the user's head and torso have an alpha value greater than or equal to 128.

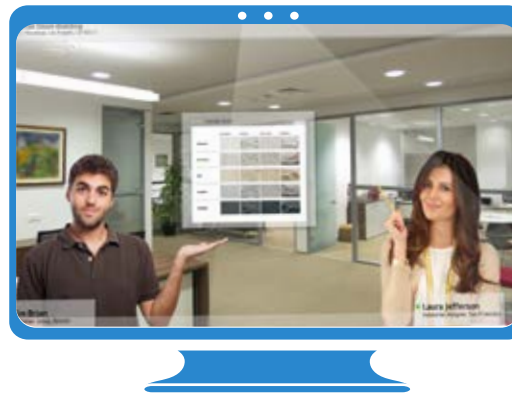
Use Cases

Some common use cases for background removal include:

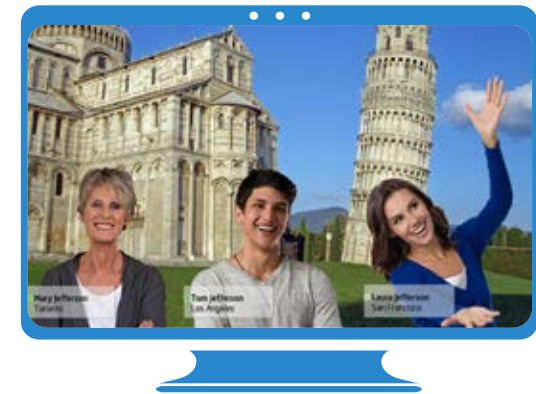
Sharing content



Sharing workspaces



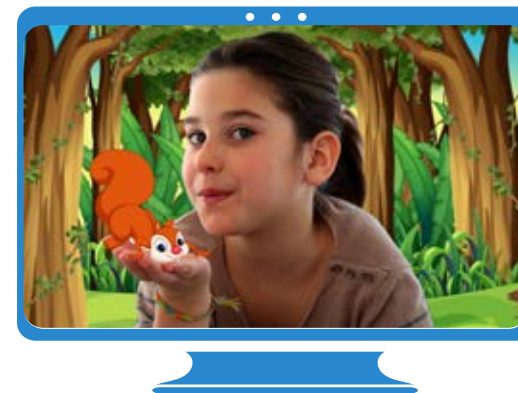
Video Chat/Teleconferencing



Photography and Video Selfies/ Enhancements



Placing the user in a different world





Best Practices✓



Utilize with good lighting conditions.

Background removal is optimized for a single user to be segmented out of the scene, but will work for multiple users if they fit into the field of view.

Don't use for privacy applications in case there is could be confidential information in the background that is being removed.

Offer the user the ability to turn segmentation on and off.

White backgrounds work best for background removal. Slight user adjustments can affect the quality of the background removal.

Play around with the visual blending effects of the edges between the segmented object and the background for different applications.

A person coming in and out of the scene might result in jagged segmentation. You can either inform the user of this or try to take care of it with visual effects.



Object Tracking

Object Tracking

The Metaio* 3D object tracking module provides optical-based tracking techniques that can detect and track known or unknown objects in a video sequence or scene. The Metaio Toolbox is provided to train, create, and edit 3D models that can be passed to various object detection and tracking algorithms.

The tracking techniques available are 2D object tracking for planar objects, feature-based 3D tracking, edge-based 3D tracking from CAD models, and instant 3D tracking.

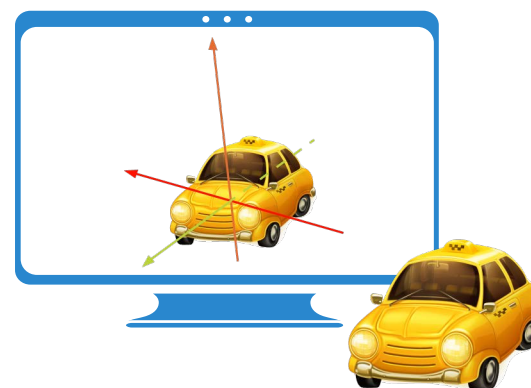
2D Object Tracking

2D object tracking is configured by providing a reference image. The algorithm tracks the image in a video sequence and returns the tracking parameters.



Feature-Based 3D Tracking

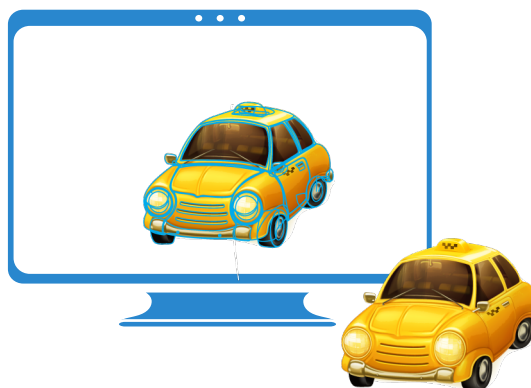
3D feature-based tracking can track any real-world 3D object. Tracking is based on a 3D feature map.



* Other names and brands may be claimed as the property of others.

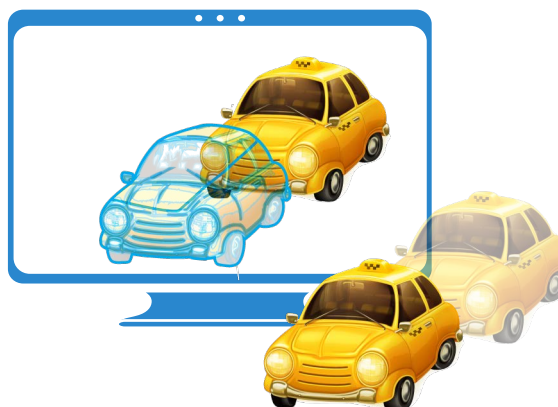
Edge-Based 3D Tracking from CAD Models

Some 3D objects are hard to detect using only feature-based 3D tracking, such as low-textured, highly reflective objects, or objects in dynamic lighting conditions. A more precise edge-based tracking can help in these cases. It uses a 3D CAD model, mesh model, or 3D-point cloud of the target object.



Instant 3D Tracking

Instant 3D tracking (also known as SLAM) allows you to create a point cloud of the scene on the fly and immediately use it as a tracking reference. With SLAM you don't need to provide any input files.





Samples



Intro

Find code samples for accessing the raw color and depths streams [here](#)



Hands

Find code samples for hand analysis algorithms (including using the 22 hand skeleton joints and gesture detection) [here](#)



Face

Find code samples for face analysis algorithms (including face detection, landmark detection, and pose detection) [here](#)



Speech

Find code samples for accessing the raw audio data, and voice recognition and voice synthesis [here](#)

