
 /IntelRealSense  
 /IntelRealSense  
[intel.com/RealSense](https://intel.com/RealSense)  
[intel.com/RealSense/SDK](https://intel.com/RealSense/SDK)

# SDK Design Guidelines

## Intel<sup>®</sup> RealSense<sup>™</sup> Camera (R200)

version 1.1

# 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: [intel.com/design/](http://intel.com/design/)\*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© 2015, Intel Corporation. All rights reserved.

# Contents

<b>Introduction</b>	<b>4</b>	<b>Mixed and Augmented Reality</b>	<b>43</b>
SDK Architecture	6	Overview	44
Camera Properties	7	Camera Tracking and Localization	45
Capture Volumes	8	Graphics Considerations	48
Camera Settings	9	Augmented Reality	49
<b>Experience Overview</b>	<b>14</b>	Virtual Reality	50
Depth-Enhanced Photo and Video	15	AR and VR with Digital Capture	51
3D Scanning	17	<b>Games</b>	<b>52</b>
Mixed and Augmented Reality	19	<b>Measurement</b>	<b>78</b>
Measurement	21	Measurement in Enhanced Photos	79
<b>General Guidelines</b>	<b>23</b>	Measurement in Scene Perception	80
Active and Inactive Camera Modes	24	<b>Face Detection and Tracking</b>	<b>81</b>
Ergonomic and Cognitive Considerations	25	Face Detection	82
Touch Interfaces	26	Avatar Control	83
Physical Environments	28	Landmark Tracking	84
Difficult Environments	29	Facial Expressions	85
Visual Feedback	30	<b>Speech</b>	<b>86</b>
General UX Design Principles	33	Best Practices	87
<b>Depth-Enhanced Photo and Video</b>	<b>34</b>		
Setup and Preview	35		
Camera Controls	36		
Controls for Editing and Sharing	37		
<b>3D Scanning</b>	<b>38</b>		
Overview	39		
Setup and Preview	40		
Scanning Controls and Feedback	41		
Controls for Editing and Sharing	42		

# Introduction

## Welcome!

With the Intel® RealSense™ SDK, you can add several new experiences to your tablet applications. These include creating depth-enhanced photos and videos, capturing 3D models of objects and people, measuring dimensions of objects and scenes, and mixed and augmented reality experiences for games and visualization.

In this section we will introduce you to the properties of the Intel RealSense Camera and to the structure of the Intel RealSense SDK.



# Intel® RealSense™ Camera Overview

The Intel® RealSense™ Cameras and SDK add depth imaging and related capabilities to a variety of computing devices. The following table summarizes the features. This document covers the Intel RealSense Camera (R200).

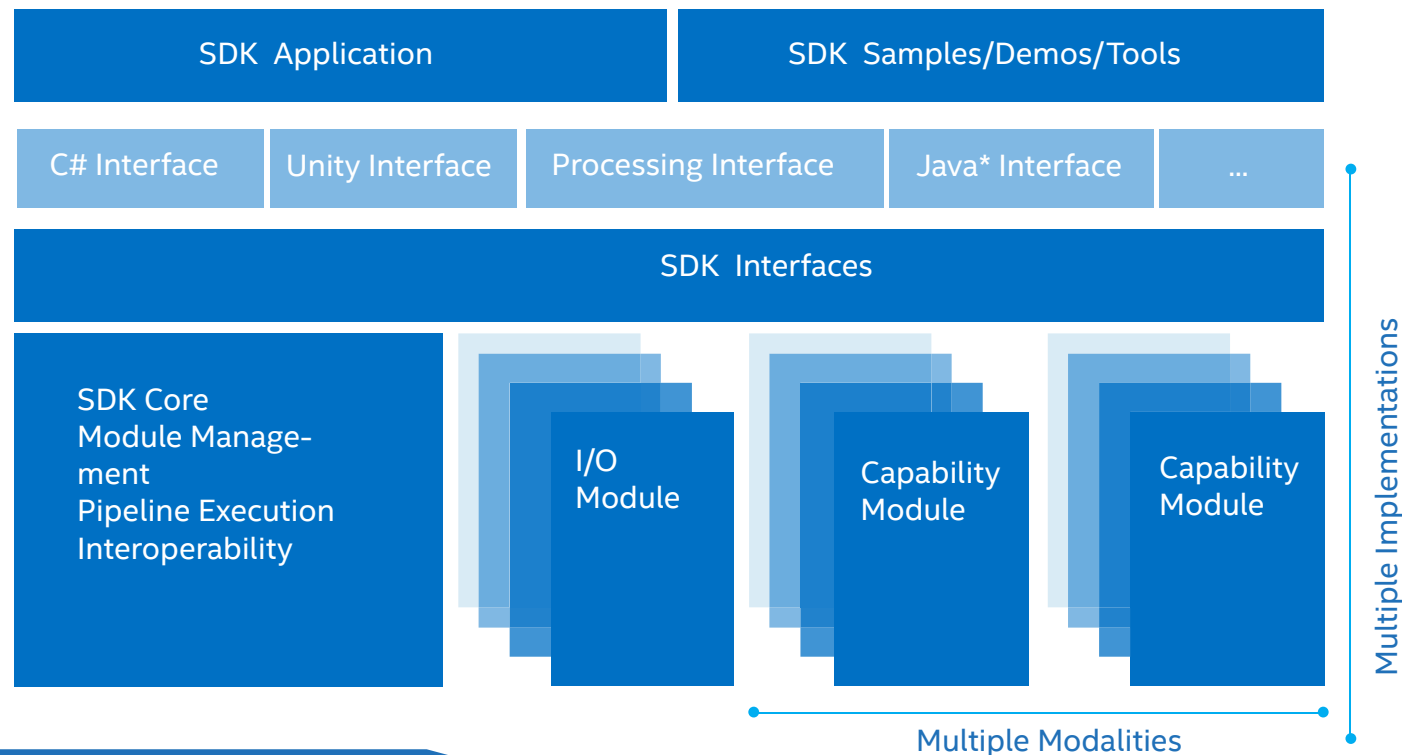
		<i>This Document</i>
UX Design Guidelines	SDK Design Guidelines: Intel® RealSense™ Camera (F200)	SDK Design Guidelines: Intel® RealSense™ Camera (R200)
Camera	F200	R200
Configuration	Short range	Longer range
Devices	Notebooks and all-in-ones	Small to large tablets (covered in this document) Developer peripheral and other configurations (not covered here)
Processor and OS	4 <sup>th</sup> Gen Intel® Core™ (Windows* 8.1)	6 <sup>th</sup> Gen Intel® Core™ (Windows 10) Atom z5 (Android* 5, Windows 10)
Features	<ul style="list-style-type: none"> <li>• Hand tracking and gestures</li> <li>• Background replacement</li> <li>• Face recognition and animation</li> <li>• Object recognition</li> <li>• Speech</li> <li>• 3D scanning</li> </ul>	<ul style="list-style-type: none"> <li>• Depth-enhanced photography and videography</li> <li>• Measurement</li> <li>• 3D scanning</li> <li>• Camera tracking and localization</li> <li>• Scene reconstruction</li> <li>• Face detection and tracking</li> <li>• Speech (Windows SDK only)</li> </ul>

# 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, such as 3D surface reconstruction, camera tracking and localization, face tracking and detection, and depth layering.

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.

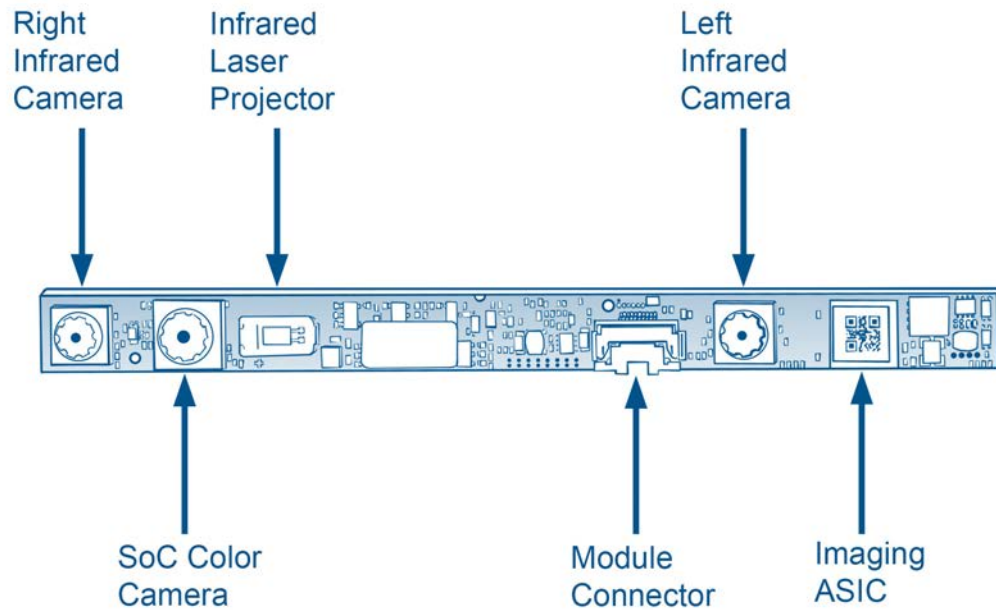
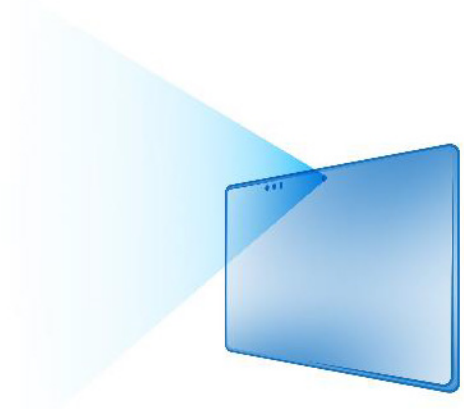


# Properties of the Intel® RealSense™ Camera (R200)

The Intel RealSense Camera (R200) captures real-time RGB with a color camera and computes depth using a system with an infrared laser projector and two infrared cameras.

The depth camera senses depth most accurately in environments that have detail and texture. Depth data will be less accurate for black or highly reflective surfaces, or for large plain flat surfaces. (See the guidelines later in this document for more information.)

The camera is mounted in tablets on the rear side, at the top of the long edge.



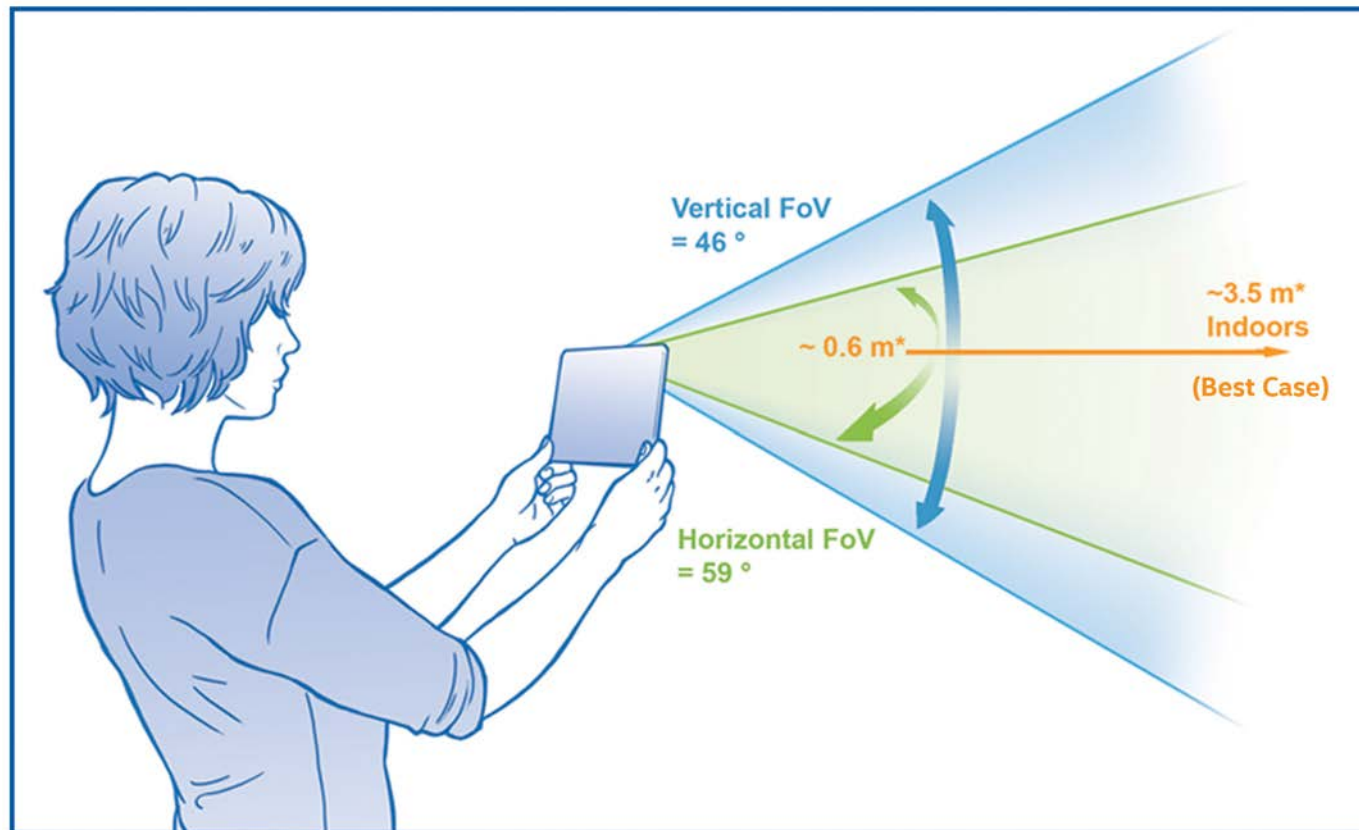
**“R200” Imaging Component Overview**

	Color	Depth (IR)
Active pixels	1920 x 1080	640 x 480
Aspect ratio	16:9	4:3
Frame rate	30 fps	30/60 fps
Field of view (D x V x H)	77° x 43° x 70° (Cone)	70° x 46° x 59° (Cone)

# Capture Volumes

Design your applications with the physical attributes of the tablet and the camera field of view in mind. The camera will be embedded in the back of tablets with sizes from 7" to 12" diagonal. Expect users to be holding the tablet in landscape orientation when actively using the camera.

The depth range of the camera extends from about 0.6 m to 3.5 m optimally. The range is shorter for darker surfaces and depends on the frame rate and other settings. For outdoors use cases, depth is reported at a greater range (up to about 10 m) with lower accuracy. Please see the SDK user documentation for more details.





# Guidelines for “R200” Camera Settings

Different camera settings will work best for different applications. In many situations, the automatic settings chosen by the SDK and middleware will work best, but there may be times when you want to set these manually.

Use the following guidelines to choose the right settings for resolution, frame rate, exposure, gain and other options.

For most situations, you should use the default settings shown on the right. In the following pages we'll show the result of changing these parameters, and how this can improve performance in some situations. The main points to keep in mind are:

- To capture fast motion, increase the frame rate or decrease the exposure.
- For outdoors, start with gain of 1 and exposure of 1, and increase exposure in small increments until you see the best results.

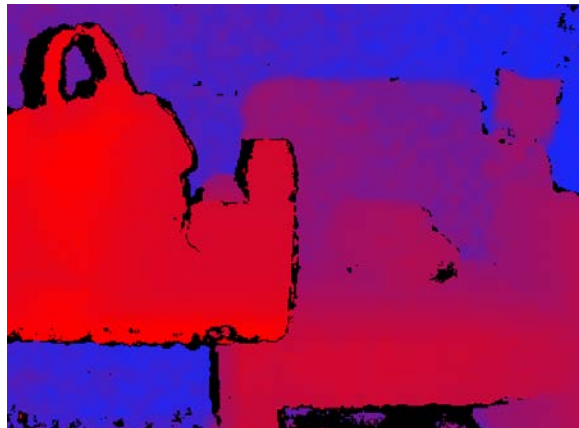
Always test the settings in scenarios that are most relevant to your application. For additional detail please see the SDK user documentation.

Best settings for most indoor applications	
Depth image resolution	480 x 360
Frame rate	30 fps
Exposure	33 ms (maximum)
Gain	1-4 (use lowest possible)

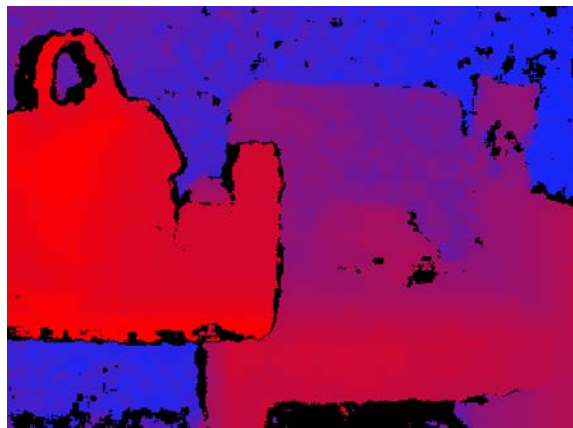
# Changing the Frame Rate and Exposure

If you have a lot of fast motion in your scene, or you expect users to move the tablet quickly, you may want to try a higher frame rate and lower exposure in order to reduce blurring of depth data across frames. The downside of this will be more holes in the depth map as shown here. Reducing the exposure will also affect how far out the camera can see; the maximum Z depth will be reduced.

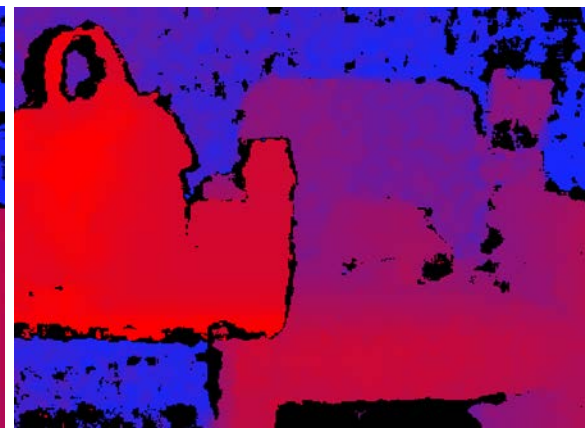
Frame rate and exposure are related – they can be set independently because the camera uses a global shutter system, but the maximum value for exposure is bound by the frame rate (these examples show the maximum exposures).



Frame rate 30 fps  
Exposure 33 ms  
(default, depth map fills in better)



Frame rate 60 fps  
Exposure 15 ms  
(more holes but better for fast motion)



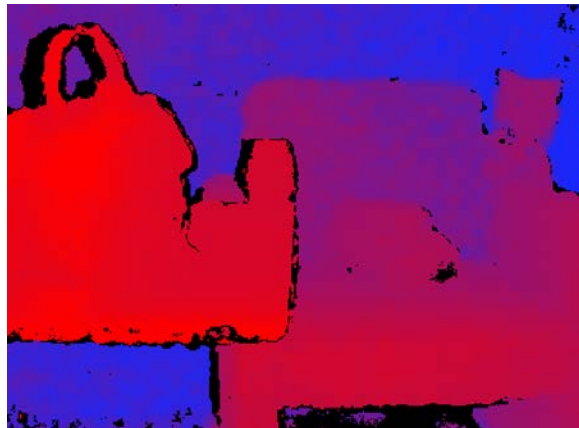
Frame rate 90 fps  
Exposure 10 ms  
(even more holes but better for fast motion)

Depth image resolution 480 x 360 (for all)

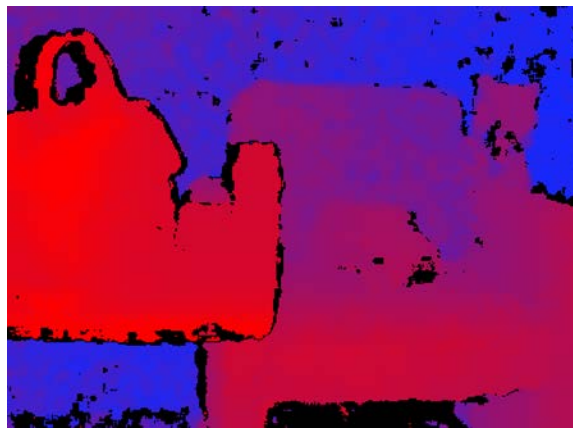
# Changing the Depth Image Resolution

Increasing resolution from 480 x 360 (the default) to 628 x 468 (approximately VGA) is convenient if you want to access the depth value at each pixel in the RGB at VGA resolution. Note that the system is simply up-sampling the depth image on the chip; there is no true increase in resolution but this will be faster than up-sampling in software.

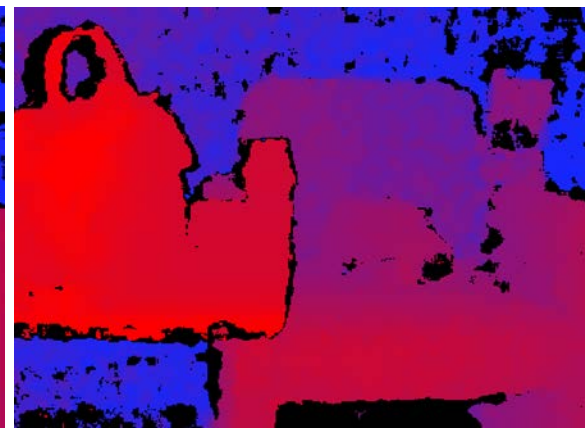
Decreasing resolution to 320 x 240 (QVGA) can be helpful because it will reduce the min-Z from about 50 mm to 30 mm. Closer objects will be visible, at the expense of an increase in depth error.



Depth image resolution 320x240  
(QVGA)  
(lower min-Z but more depth error)



Depth image resolution 480x360  
(default)



Depth image resolution 628x468  
(approx. VGA)  
(depth image upsampled for  
convenience)

Frame rate 30 fps, Exposure 33 ms (for all)



# Changing the Gain

Try the auto gain setting for most situations. If you set it manually, choose the lowest gain where depth results are reported in the range that you want. As you increase gain, further surfaces will come into range and nearer surfaces will begin to saturate (e.g. as in the gain 3 and 4 examples below).



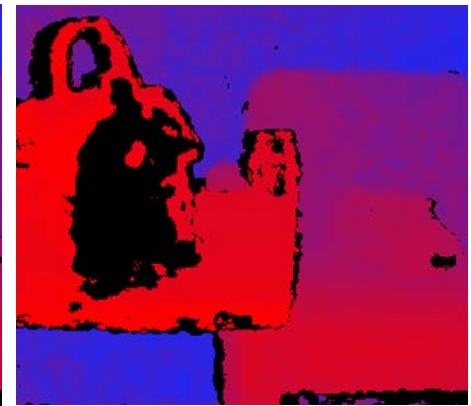
Gain 1  
(too low for distant surface)



Gain 2  
(about right)



Gain 3  
(too high for near object)



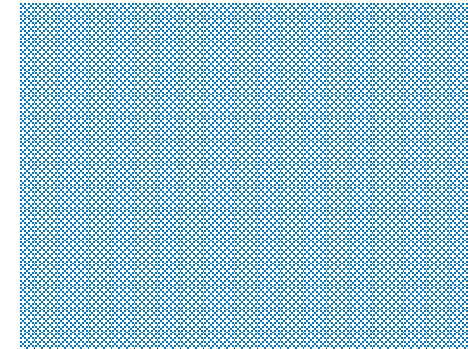
Gain 4  
(too high for near object)

Depth image resolution 480x360, frame rate 30 fps, exposure 33 ms (for all)

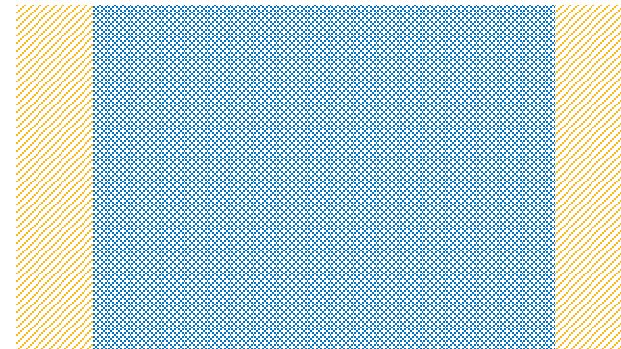
# Camera Settings – Other Considerations

## Color Image Resolution and Aspect Ratio

- These are largely your choice according to the desired design and performance for your app – there are no trade-offs here with depth quality or range.
- Choose 1920x1080 for a widescreen (16:9) color image that will best match most tablet aspect ratios, but be aware that depth is only capture in the center region of the image at a 4:3 aspect ratio. See the illustration on the right for a comparison.



With color resolution 640x480, the depth and color images fully overlap



With color resolution 1920x1080, depth overlaps only the center region



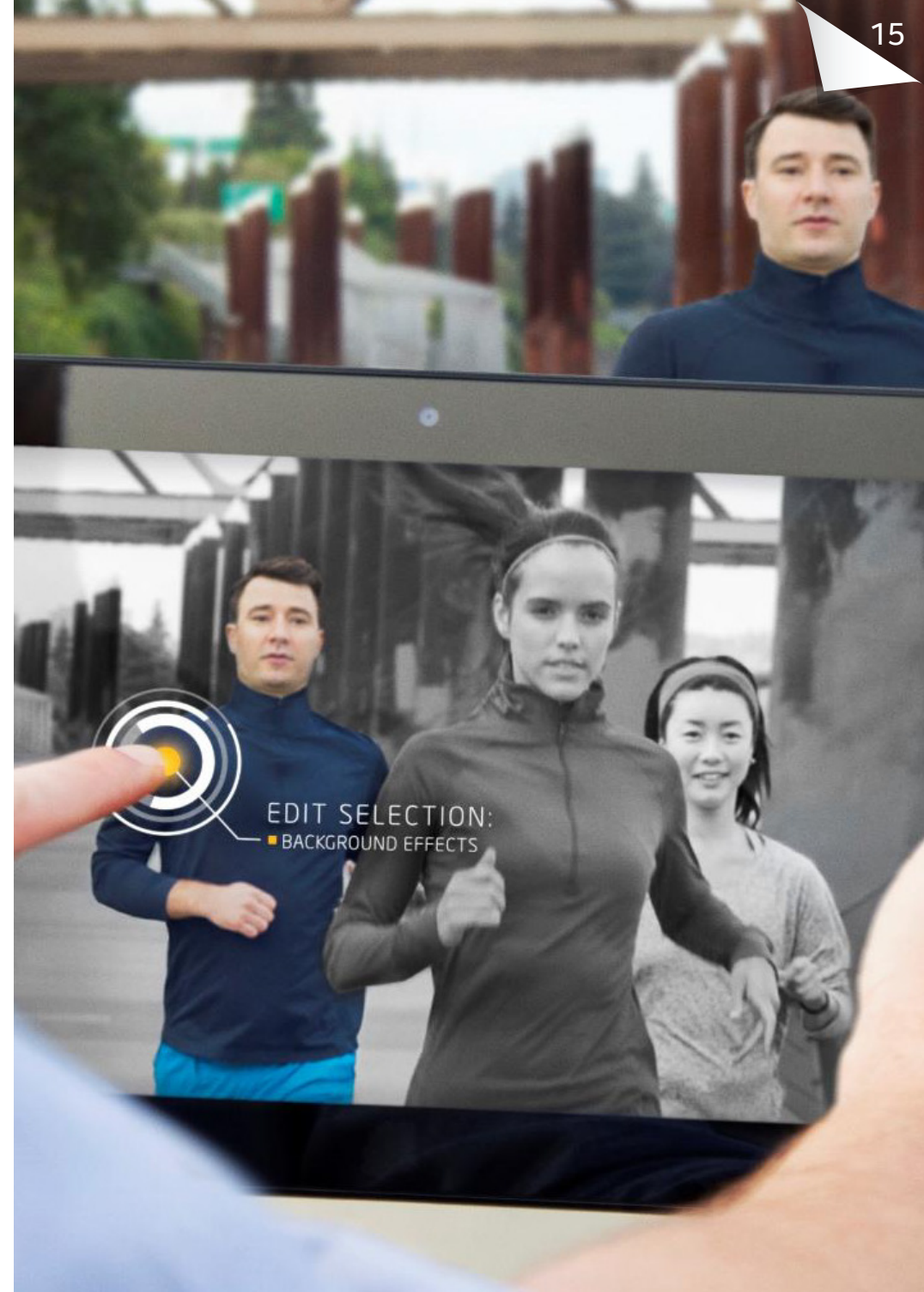
# Experience Overview

The Intel® RealSense™ SDK enables several new experiences you can employ in your applications, individually or together. This section gives an overview of these experiences. Later sections describe design guidelines and best practices for each one in more detail.

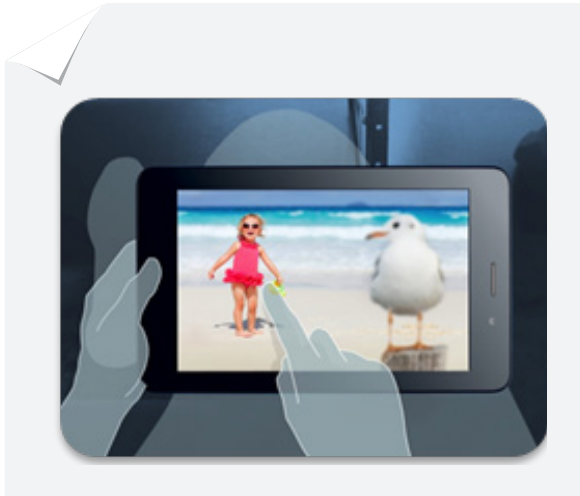
# Depth-Enhanced Photo and Video

Intel RealSense Cameras capture layers of the scene that can be separated (or “segmented”) according to depth. This allows for applying visual effects to specific layers — for example, a grayscale or blur effect. Parallax and dolly-zoom motion effects are also possible.

These applications can be implemented using the Enhanced Photography and Videography module of the Intel RealSense SDK.

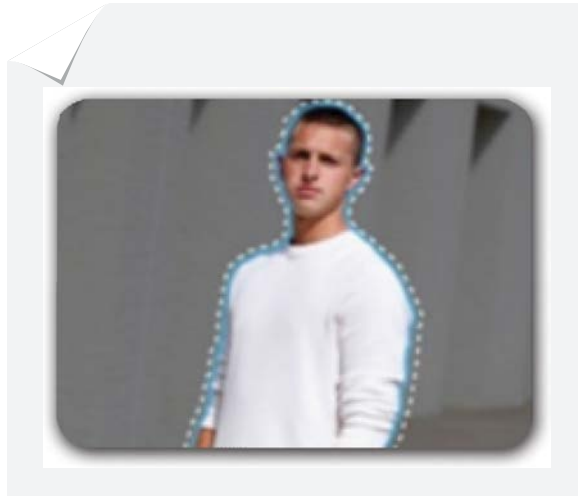


# Use Cases for Depth-Enhanced Photo and Video



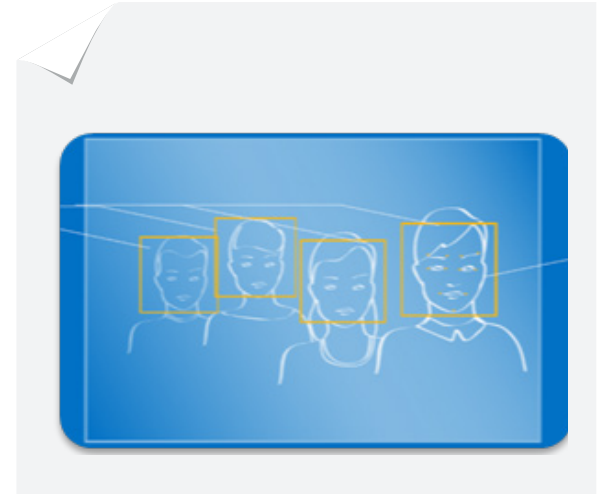
## Artistic effects

- Apply effects or paint using depth
- Selective blurring of foreground or background
- Interactive parallax and zoom



## Segmentation

- Depth-assisted selection and segmentation of people or objects



## Face Detection and Tracking

- Track and augment faces in enhanced videos
- Create avatars that mimic the user's face



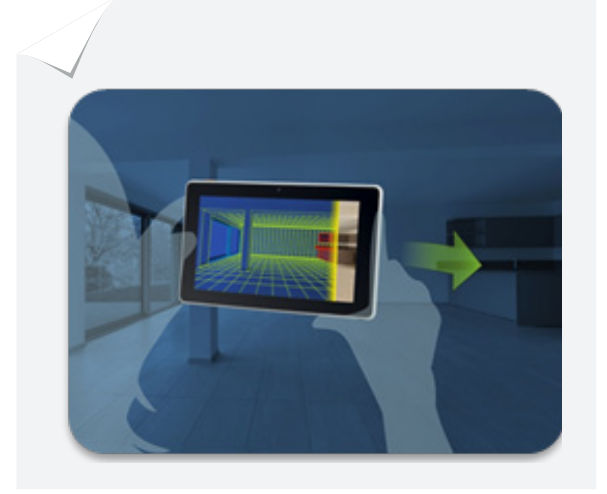
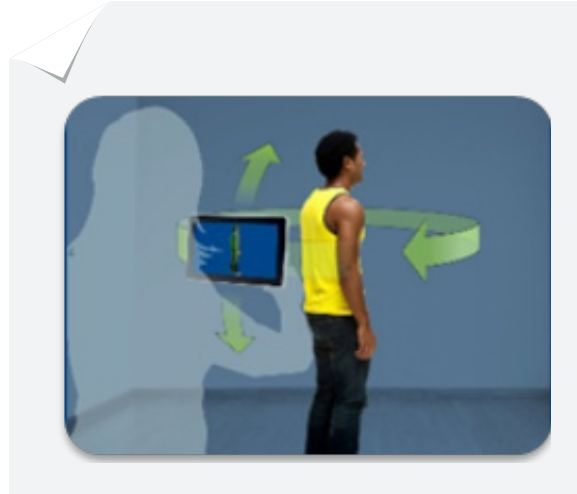
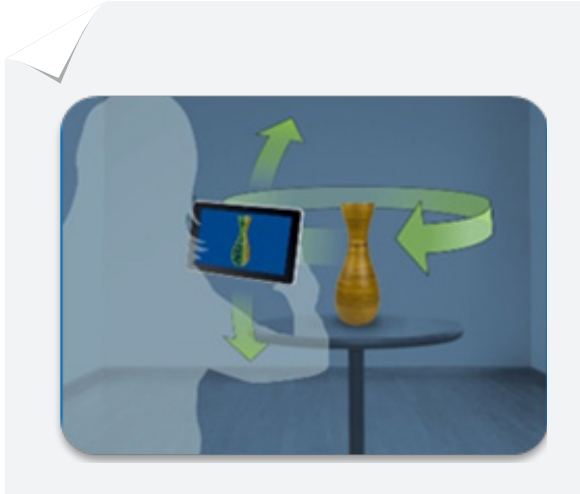
# 3D Scanning

3D scanning is about digitizing objects, people, or scenes, in order to export them for 3D printing or sharing. The 3D scanning module provides functions for capture, editing, and exporting that you can use to build applications focused on the capture experience.

3D scanning can also be implemented as part of augmented reality or game applications, using the Scene Perception module. Scene Perception is optimized for real-time processing and display of meshes, whereas 3D capture is optimized for capturing high-quality meshes, as well as editing and exporting them.



# Use Cases for 3D Scanning



## Capture for 3D printing

- Capture and edit models of objects or people for 3D printing

## Capture for sharing

- Capture models of objects or people and post to social media or e-commerce sites

## Capture scenes

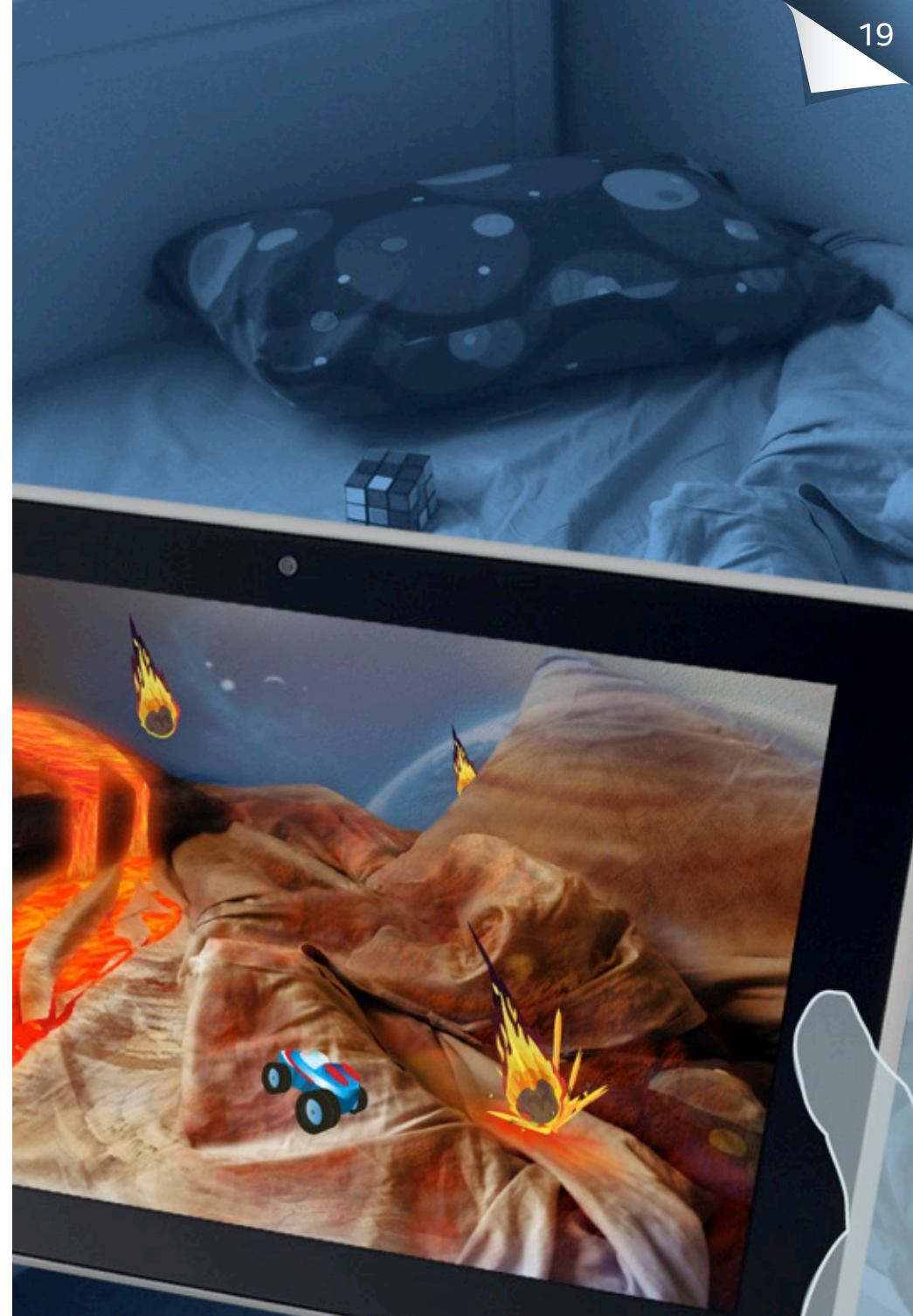
- Capture a model of a room and create a virtual walkthrough video

# Mixed and Augmented Reality

Mixed reality is about displaying and interacting with digital and physical content in a blended virtual world. With the camera and the SDK you can create a variety of mixed reality, augmented reality, and virtual reality applications based on the following capabilities:

- Real-time tracking of the tablet camera's position and orientation in the world. Tracking allows you to render images on the screen that mimic a "magic window" into a virtual scene that updates as the user moves the tablet. "Localization" allows tracking to resume quickly in a new session within the same environment.
- Real-time reconstruction of geometric meshes that represent the surfaces in the scene.
- Rendering of virtual content blended realistically with the physical world. Virtual objects or annotations can "stick" to places in the real scene, and can be drawn in front or behind of real objects with correct occlusion and shadows. Game characters can hide behind real objects.

Mixed and augmented reality experiences can be created with the Scene Perception module of the SDK.



# Use Cases for Mixed and Augmented Reality



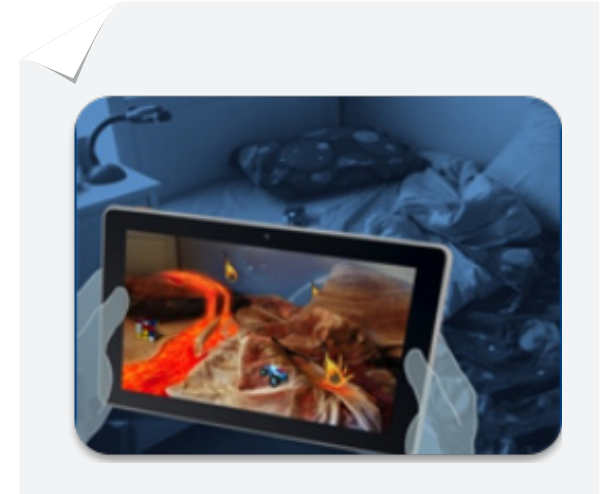
## Visualization

- View virtual objects like furniture and art, drawn accurately in your physical environment



## Education

- View information as annotations in augmented reality



## Games

- Augmented reality games set in your environment
- "Capture and play" games that integrate real objects into a virtual reality game



# Measurement

Depth cameras measure a distance at every pixel from the camera to the corresponding point in the scene. In addition, the computer vision modules in the SDK allow for measurement between arbitrary locations in the scene. You can implement three styles of measurement:

- Measuring distances between two points in a captured photo — using the Enhanced Photo and Video module.
- Measuring distances between two points in a captured 3D model of an object or scene — using the 3D Scanning module.
- Measuring distances between two points in a 3D scene in real-time (without first capturing the scene), using the device as a kind of virtual tape measure — using the Scene Perception module.



# Use Cases for Measurement



## Measuring Distances

- Take a depth photo or capture a 3D scene and then select points in the images to record the measurements

## Measuring Dimensions

- Measure and label object dimensions when posting items for sale online

## Virtual Electronic Tape Measure

- For home redecoration in larger spaces



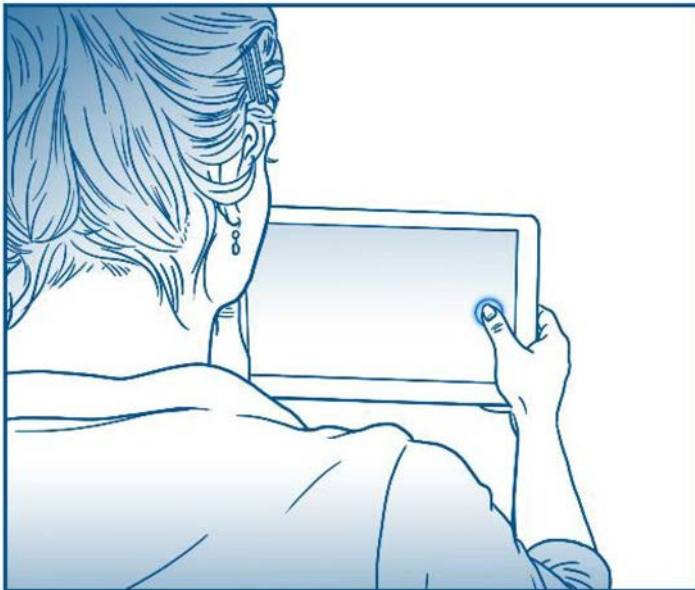
# General Guidelines

In this section we discuss general principles to consider when designing applications for the Intel® RealSense™ Camera (R200). In the subsequent sections we cover detailed guidelines in specific feature areas.



# Active and Inactive Camera Modes

It makes a big difference to the user whether they are holding up the tablet to look at a view through the camera or are holding the tablet in a relaxed position to interact with content. Throughout this document we will refer to these two modes of user operation:



**Active Camera** means that the camera is on and the user is viewing and potentially interacting with the scene in real time. Users will hold the tablet with one or two hands, upright as though it's a viewport or "magic window." Their ability to touch the screen comfortably is limited.



**Inactive Camera** means that the camera is off. Users will be holding the tablet to interact offline with content – for example, editing a photo or scene just captured. Since the tablet is not a viewport, users are free to set the tablet on a table or their lap for comfort. Users can more comfortably touch anywhere on the screen.



# Ergonomic and Cognitive Considerations

When designing your applications keep in mind the physical and practical nature of using a tablet with a camera: tablets are mobile, heavier than phones, and made for casual, short interactions compared to traditional PCs. Consider the following recommendations:

- **Keep Active Camera interactions short.** Holding a tablet for long periods can be tiring and even painful for some users. Don't require users to hold the device in active camera mode for more than a few minutes. Break up these active-camera tasks into chunks that can be interrupted. Allow editing and adjusting tasks to be done in inactive camera mode whenever possible. Let people do tasks in different orders if they choose.
- **Design for interruption.** As with any mobile device apps, expect users to be interrupted – they might pause during a step and then come back to it, or they might close the app and reopen it. Aim to allow users to pause any task and pick up where they left off without losing work.
- **Minimize text entry.** It's easier to select things than to type them on a mobile device. Don't require any text entry during active camera mode.



# Touch Interfaces: Placement and Basic Controls

Follow best practices for touchscreen interfaces and pay particular attention to the placement of controls for the different camera modes and postures:

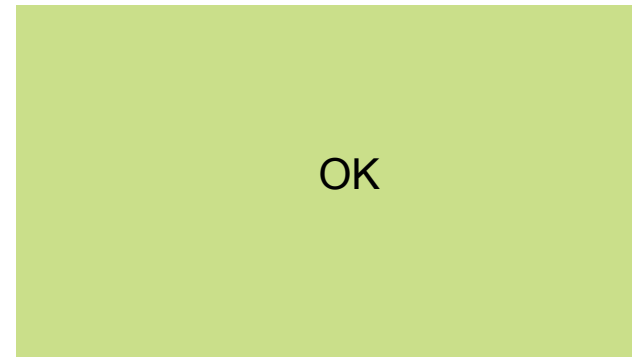
- For **active camera mode**, stick to one-touch controls, reachable by an index finger or thumb. (There are some exceptions for games that we describe later.) Use buttons and other single-action controls. Avoid touch-and-hold or touch-and-drag interactions.
- For **inactive camera mode**, the user is able to rest the tablet, so it's okay to include multi-touch controls. It's also okay to use continuous interactions like touch-and-hold, touch-and-drag, or pinch zoom.

Keep familiar controls in consistent and expected positions. Primary actions like a camera capture button, or a “scan” or “play” button should generally be on the right side of the screen. Secondary actions for selecting a mode or a tool, should generally be on the left. (But you may consider a user option to mirror these controls for left-handed users.)

***Make the main controls easy to reach with the thumbs during Active Camera modes .***



***During Inactive Camera modes, placing controls elsewhere is acceptable.***



# Touch Interfaces: 3D Manipulation and Viewing

Many camera applications involve manipulating and viewing 3D content. Here are some best practices for implementing these interactions.

**Selecting in 3D** - To select an object or surface in a 3D scene, the two most common techniques are:

- **Touch to select** the object directly. When the user touches a point on the screen, the application should cast a ray into the scene to determine the closest object it intersects with. This technique will be easiest for users to understand. Use it for most situation.
- **Aim to select** with a reticle view. Draw a reticle in the center of the screen to help with aiming. The user moves the tablet to aim at a target and then touches a “select” or “fire” button at the side of the screen to make the selection. This technique will be more difficult for some users to understand, but it can be more efficient and comfortable in games if the user has to perform it frequently.

Note that aim to select only applies in active-camera mode. For inactive-camera mode, touch to select should always be used.

To move objects in 3D space, use the select interaction with an additional “drag” phase.

**Viewing in 3D** - During active-camera mode, the view is tied to the tablet position and orientation. During inactive-camera mode, you may want to give the user the option to change the view, using standard 3D viewer controls. Allow basic orbital rotation and zooming, and constrain the degrees of freedom where appropriate. For example, when rotating an object, often the user only wishes to rotate about the vertical axis.

Refer to the references later in this section for more best practices when designing 3D interfaces.



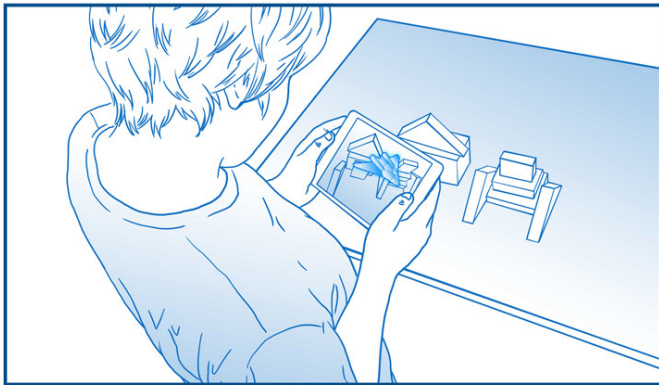
# Physical Environments

## Space of interaction

- Plan for the space where users will interact with your application. What objects or scenes will users capture or augment with digital content? How much empty space is needed? The illustrations here show three common scenarios — small and large indoor spaces, and outdoor spaces.

## Physical props

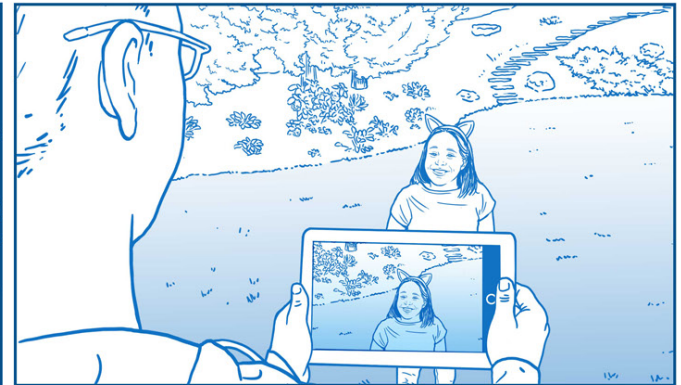
- What objects will users need to have available when they use with your application? Mixed-reality games make use of physical toys or other objects that can be captured and inserted into the virtual scene. Ensure that your game's requirements are reasonable and that you have fall-back plans so that users can always do something interesting, even if they don't have props available.



Small space – table tops and floors. About 0.5 to 2 meters across. For mixed reality and 3D capture. Particularly for games.



Large space – room scale. About 2-4 m across. For mixed reality 3D capture.



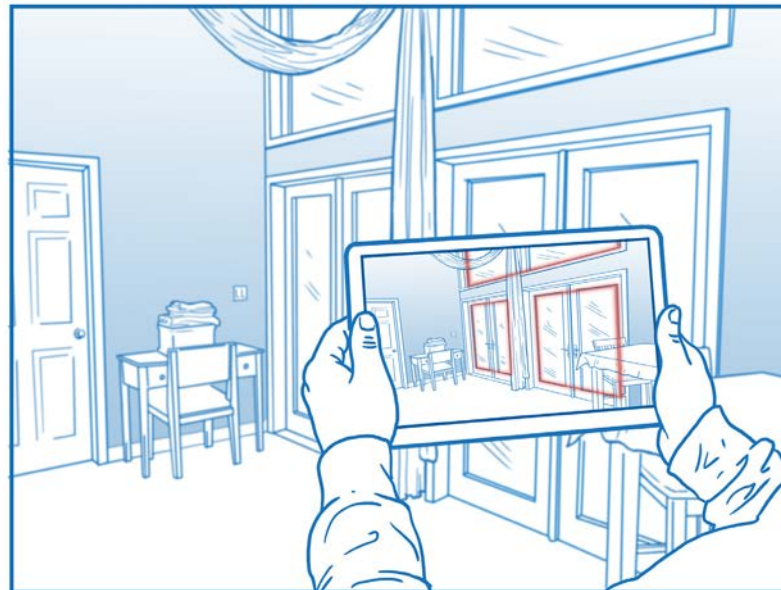
Outdoor space – everything in view of the camera. For depth-enhanced photo and video experiences. (Note that depth is measured over a limited distance.)

# Dealing with Difficult Environments

The camera may have difficulty capturing certain materials and scenes. These include surfaces that are black, highly reflective, or extremely bright (such as sun coming through a window). Large blank featureless walls can also be difficult for the camera to capture. Avoid highly reflective or semi-transparent virtual objects.

Design for this by

- Avoiding use cases that are likely to fail. For example, don't design an app for scanning glass objects or measuring windows.
- Detecting difficult situations and informing the user. Alert the user if no surfaces are being seen within range, and use the depth quality information provided by the SDK to flag problem areas in the scene. This helps the user know when to move elsewhere or make the scene more suitable (for example by putting more objects in a play space for a mixed-reality game). For depth-enhanced photo and video applications it is particularly important to indicate that the subjects are in range and that accurate depth readings are available.
- Fail gracefully by ignoring problem areas. Chances are there will always be some reflective or dark "holes" in a scene. Use the hole-filling features in the SDK modules to cover these, or make the holes a unique part of the design in mixed-reality games.

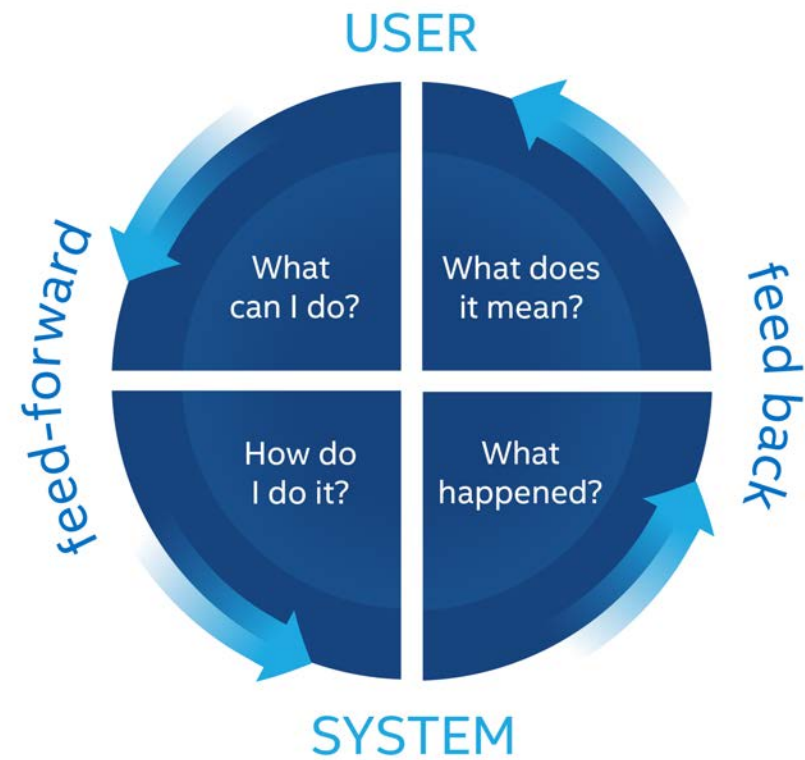


# Visual Feedback

Your applications should always keep users informed of what is happening with the system and what it means — this is the basic principle of feedback. The counterpart to feedback is feed-forward, which tells the user what they can do, and how to do it.

Locate visual feedback where the user is looking – at an object of interest, or near the controls you expect them to be looking at. “Status bar” style feedback at the top or bottom of the screen is generally less effective than feedback closer to the center of the display.

Make text and feedback visuals legible and clear against different types of scenes. During active-camera mode, users will have to see the feedback clearly against a live video feed. In general it's best to use text in light colors backed by a dark border or shadow.



Don Norman's user action framework



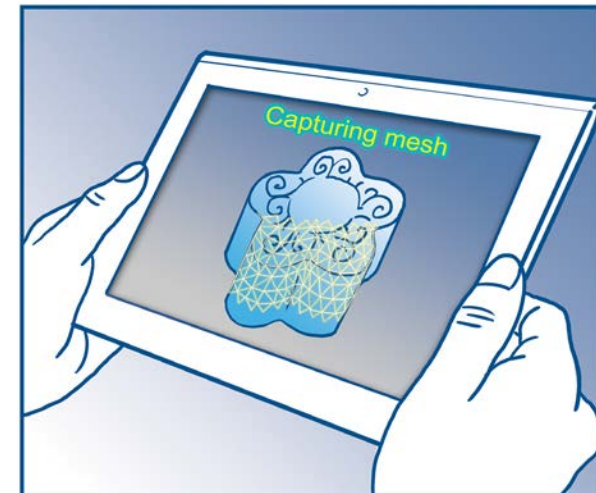
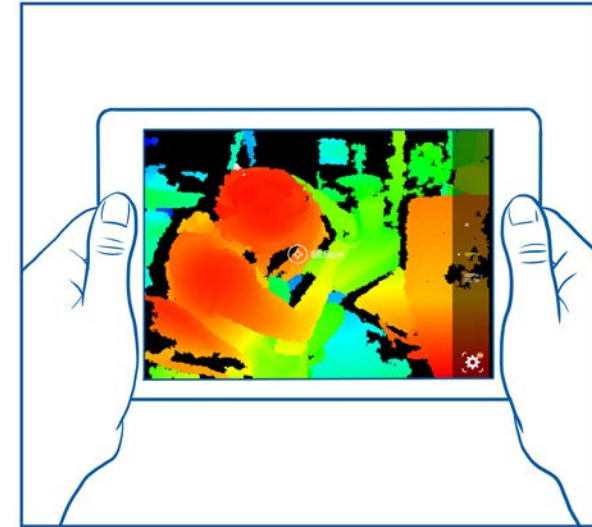
# Visual Feedback: Depth Images

Depth images are an important diagnostic tool that you'll use often as you develop applications (see the example at right).

But in most circumstances, users won't need to see depth data in this raw format. This is true for most mixed-reality games. Instead show a representation of what's been done with the depth data — such as creating a mesh to represent a surface.

In some situations, however, **depth images** with a color gradient can be useful to help users accurately perform tasks, such as with enhanced photo and video tasks where understanding the relative depth of surfaces can be helpful.

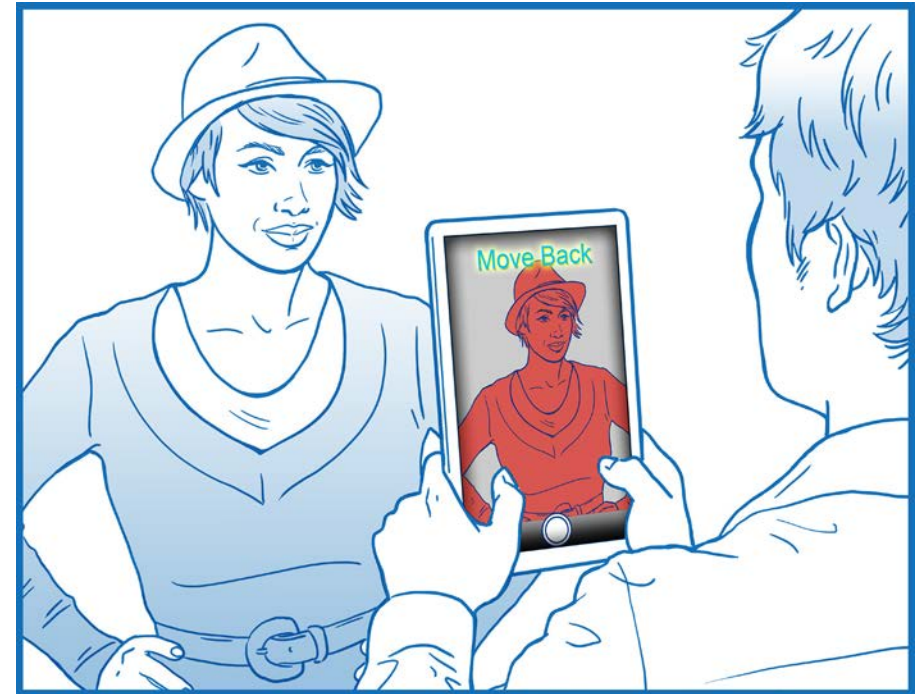
In these cases, draw the depth as a semi-transparent overlay on the camera's RGB feed. See the SDK samples for easy ways to implement this.



# Visual Feedback: Range Alerts

Show range alerts for

- Helping the user find the right distance from the object of interest. Show “move closer” or “move back” instructions or a mesh to highlight the area that’s being captured. For enhanced photo and video applications, inform the user if nothing is being captured within the usable depth range.
- Directing the user back to the scene to resume tracking. This is best done in combination with a depth image overlay. The alerts can be arrows or a text instruction like “Please move back to the last recognized location.”
- Warning the user if they’re about to go out of range of the scene. This can be helpful in mixed-reality games.





# General UX Design Principles

The following highlights some common UX advice that we feel is particularly important to remember when designing applications using the Intel® RealSense™ Camera (R200).

**Give the user control and be responsive.** Ensure that any computationally intensive processing happens in small chunks and can be paused or interrupted by the user. Show progress updates and be responsive at all times.

**Anticipate and forgive errors, and allow for retries.** Expect users to accidentally touch the wrong button, so provide an undo function when possible. Let users redo complex tasks like 3D capture, so they can easily try for a better result.

**Practice user-centered design.** Involve real users in your development process. Conduct usability tests to find issues and improve your design. Guidelines are not a substitute for user testing.

There are many strong resources about designing and developing successful user experiences. These are some of our favorites:

## General UX design and research

Don Norman. *The Design of Everyday Things, 2nd Edition*. 2013.

Jeffrey Rubin and Dana Chisnell. *Handbook of Usability Testing*. 2008.

## 3D graphics, Mixed reality and 3D user interfaces

Hughes, van Dam, McGuire, Sklar, Foley, Feiner, and Akeley. *Computer Graphics: Principles and Practice, 3rd edition*. 2013.

Bowman, Kruijff, LaViola, Poupyrev. *3D User Interfaces: Theory and Practice*. 2005.

Alan Craig. *Understanding Augmented Reality*. 2013.

## Computer vision

Greg Borenstein. *Making Things See: 3D vision with Kinect, Processing, Arduino, and MakerBot*. 2012.

## Games

Jesse Schell. *The Art of Game Design: A Book of Lenses, 2nd Edition*. 2014.

## Touch and mobile interfaces

Dennis Wixon and Daniel Wigdor. *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. 2011.

Steven Hoober and Eric Berkman. *Designing Mobile Interfaces*. 2013.



# Depth-Enhanced Photo and Video

This section describes best practices for designing experiences that leverage depth data for creative photo and video applications.

# Depth-Enhanced Photo and Video: Setup and Preview

To ensure successful enhanced photo and video experiences that apply depth filtering techniques, include the following elements in the initialization of the app:

- Encourage the user to select indoor or outdoor scenes that are well lit and that have interesting depth variation within the camera range. Show a live depth/distance indicator to help the user.

Detect or warn about difficult surfaces that won't be captured well. See section: [difficult scenes](#).

- The main part of your experience will be a live preview, similar to any camera application. Include the above features and warnings within the live preview as text overlays.



# Camera Controls

Make the capture button prominent and easy to touch with a thumb. For video interactions, use the same button to start and stop recording. Include audio and visual feedback. Use standard practices such as turning the button red when recording is active.

After a photo or video is captured, show immediate confirmation. Animation can be helpful to indicate the photo or video was stored and how the user can access it (e.g., show the photo zooming out to a tray).

These interactions happen in an active-camera posture, so be sure to limit any other controls. If controls are available to preview the depth-enhanced filters, make them simple buttons in easy-to-reach zones at the sides of the screen.



# Controls for Editing and Sharing

After the user has captured a photo or video, give them the ability to edit and share the images in an offline (inactive-camera mode) phase.

The interface for this phase should generally use a gallery or tray view that shows prior images and a toolbox of controls. Your application should normally start in this mode to help orient the user.

Editing/enhancement operations should generally proceed as follows:

1. The user selects an operation to be performed, such as a filter to be applied. This enters the system into a mode (for example “blur mode”).
2. The user indicates on the image where to apply the operation.
3. The user presses a “done” button to indicate completion and to exit out of the editing mode.

Provide an Undo button so that users can reverse any of these operations. If possible, use **nondestructive editing** – keep the original image so that the user can go back to it at any time.

Sharing operations, such as sharing the photo to social media or emailing should follow common usage patterns for mobile devices. Be aware of image size restrictions when exporting images and warn the user about any potential loss of data.





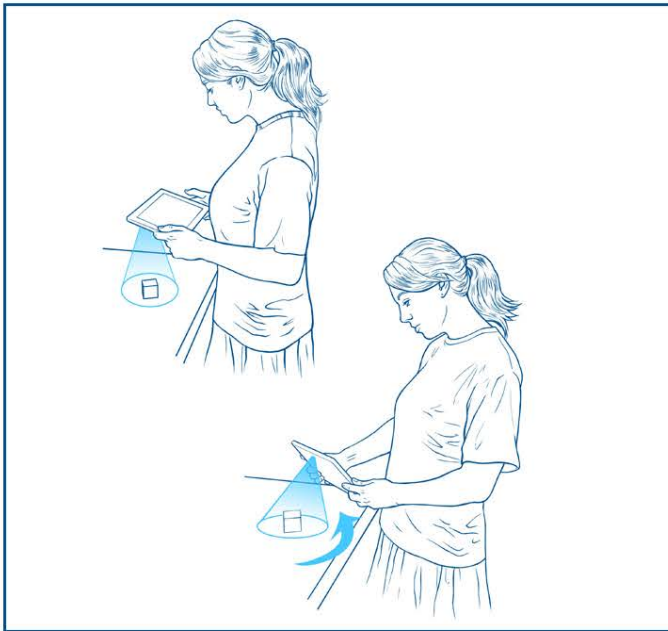


# 3D Scanning

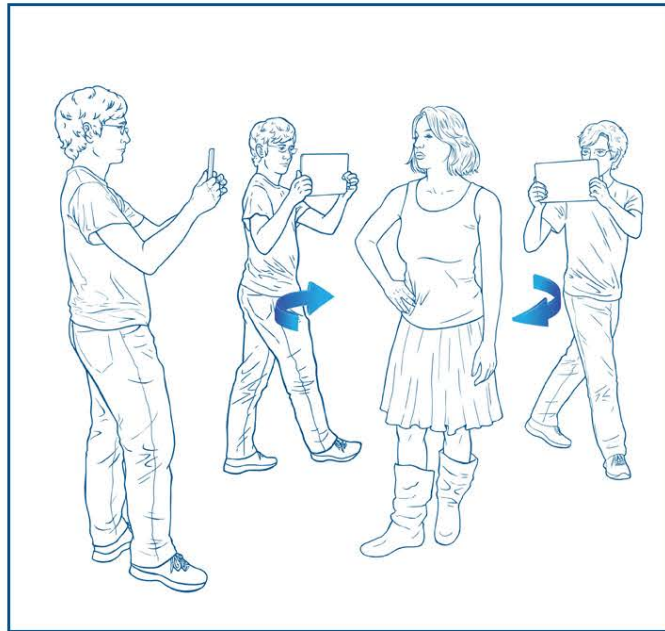
With the 3D Scanning module of the SDK you can build applications for capturing objects, people, and environments. To help users achieve satisfying results for 3D printing or sharing, follow the design guidelines in this section when building your app.

# 3D Scanning Overview

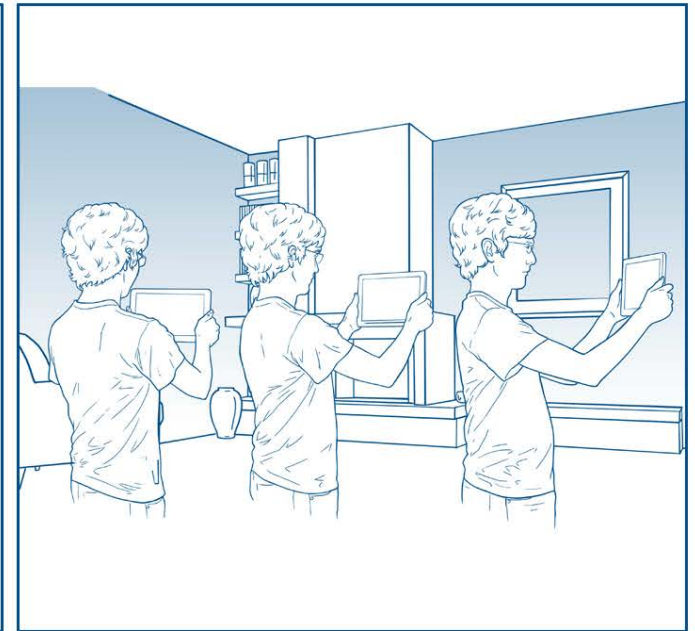
Focus your 3D scanning application on one of the three scenarios described below, as the interaction flow will differ according to the size and type of surfaces being captured. You will also need to choose an appropriate voxel resolution in the SDK, according to the scenario.



**Small object capture:** create a model of a small object (about 20-50 cm). You'll need to instruct users to place the object on a flat surface and move around the object to capture it from side and top views.



**Person capture:** create a model of a person's head, torso, or full body. You'll need the subject to remain still while the user walks around them and captures side views.



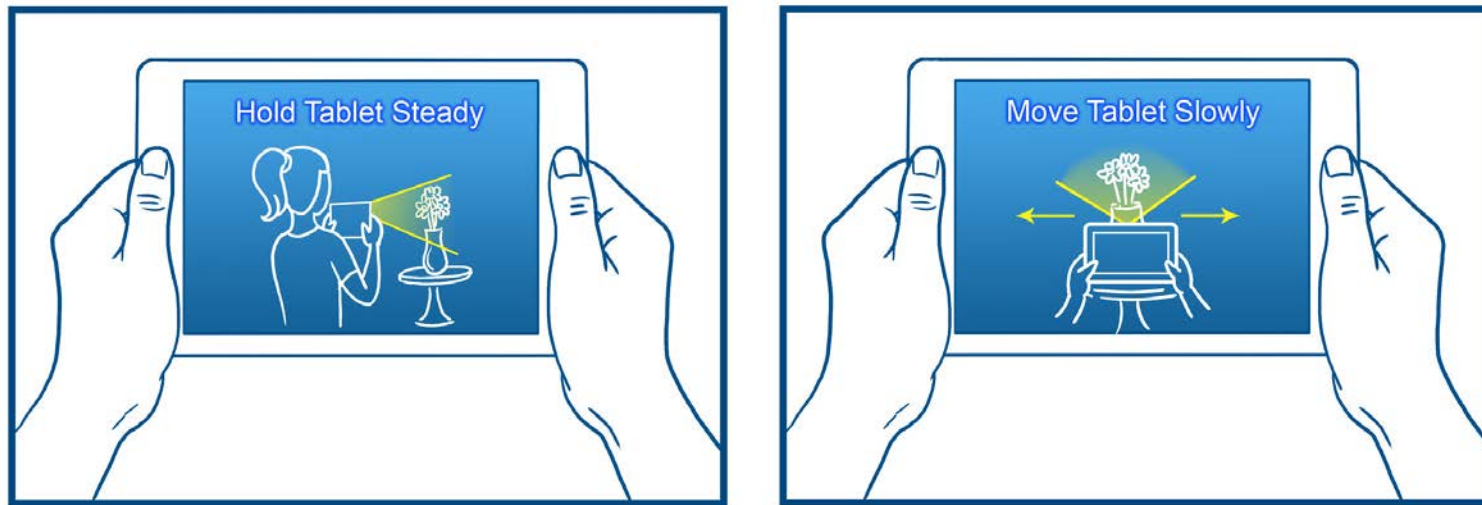
**Scene capture:** create a model of the room the user is in. The user will scan the tablet around the room as if taking a panoramic photo.

# 3D Scanning: Setup and Preview

Give simple instructions to start. Show users a simple drawing or animation to communicate the process — don't use text alone for instructions. The drawing should show the user, the tablet, and the subject of scanning. Use arrows or motion to indicate how the user should move in order to begin and continue the capture process.

Detect and avoid problem scenarios. Alert the user if no surfaces are seen within the usable depth range, or if the depth quality reported by the SDK is poor. If there are “holes” caused by dark or reflective surfaces, indicate these in the live preview so that the user can adjust the scene if they wish.

Preview the subject or area that will be captured. For an object or person, highlight the corresponding surface. For a scene, show a bounding box to indicate the region that will be scanned, and if appropriate, allow the user to adjust the size of this region.





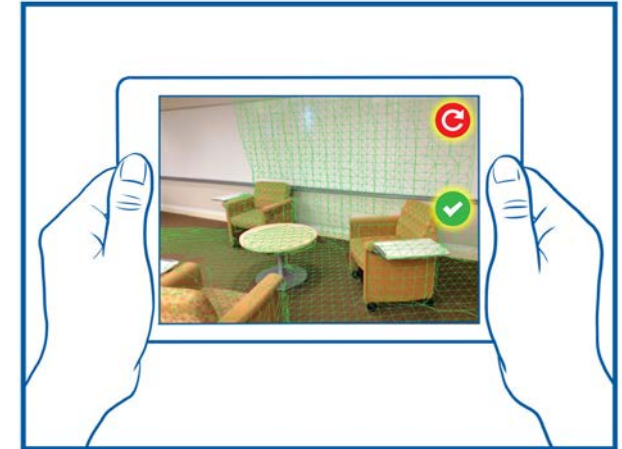
# 3D Scanning Controls and Feedback

During the scanning process, the user will need to move the device around the object or scene while viewing a live preview. Make the capture button prominent on the side of the screen. Use the same button for starting as for stopping capture (but change its label and color).

During capture it's essential to communicate

- **Activity** - show that capture is happening. Give immediate feedback, such as a simple wireframe mesh to show that the surface is being seen and processed.
- **Range alerts** - indicate to the user if they are out of the depth range of the camera (either too close or too far from the surface). New users may try to scan very close; if no depth data is seen suggest that the user try moving back.
- **Quality** - preview the mesh as it's being constructed so that users can notice errors and situations when they need to restart. The real mesh can be drawn at a slower rate than the "activity" feedback described above.
- **Coverage and gaps** - help users know where there are holes in the mesh, and when the mesh is complete, so that they will know when they are done.
- **Motion** - use simple text prompts (such as "move to the right") or visual indicators such as arrows. For some captures, you will want users to complete a circle around the person or object — show this graphically, such as with a circle that surrounds the subject and changes color as different portions are completed.
- **Speed** - Try not to require your users to move unnaturally slowly. If you do need to ask users to slow down, the most effective feedback is simple text, or a speed indicator using velocity measured by the device's IMU.

If possible, allow users to stop and start multiple times during a single capture. Breaking up the task into chunks helps reduce fatigue and errors. During the breaks, allow the user to view the mesh using an inactive-camera view. Seeing partial results in this way will help users know that they're getting the desired results. For long captures (taking more than about a minute), it's especially important to let the user complete the capture in several sessions.



# Controls for Editing and Sharing

For the editing phase of 3D scanning, use a layout and controls similar to those offered in depth-enhanced photo and video applications (see the guidelines earlier in this document). Your main screen should be a gallery or tray view of previously captured models that users can perform actions on – for example: redo capture, edit model, share model. After the user completes a capture they should also arrive at the gallery screen, where they can choose whether to edit and share the just-captured model, or capture a new one. Splitting the capture and edit steps is recommended so that users have the most flexibility in how they do their work.

Keep in mind that capturing happens in active-camera mode, while editing and sharing happens in inactive-camera mode, where more complex controls are usable.

The following types of editing features can be implemented with the SDK and users will need these features in order to produce satisfactory 3D models for sharing or printing.

- Floor removal to cleanly separate the model from surrounding surfaces.
- Hole filling or solidification, to cover errors or prepare the model for 3D printing.
- Cropping of extra surfaces or errors.
- Simple smoothing and mesh simplification, to improve appearance or reduce file sizes.
- Texture mapping. Users will expect color textures on all models.

Other recommendations:

- Provide for easy integration with existing 3D file formats and tools.
- Provide measurement and annotation features within 3D scanning apps.



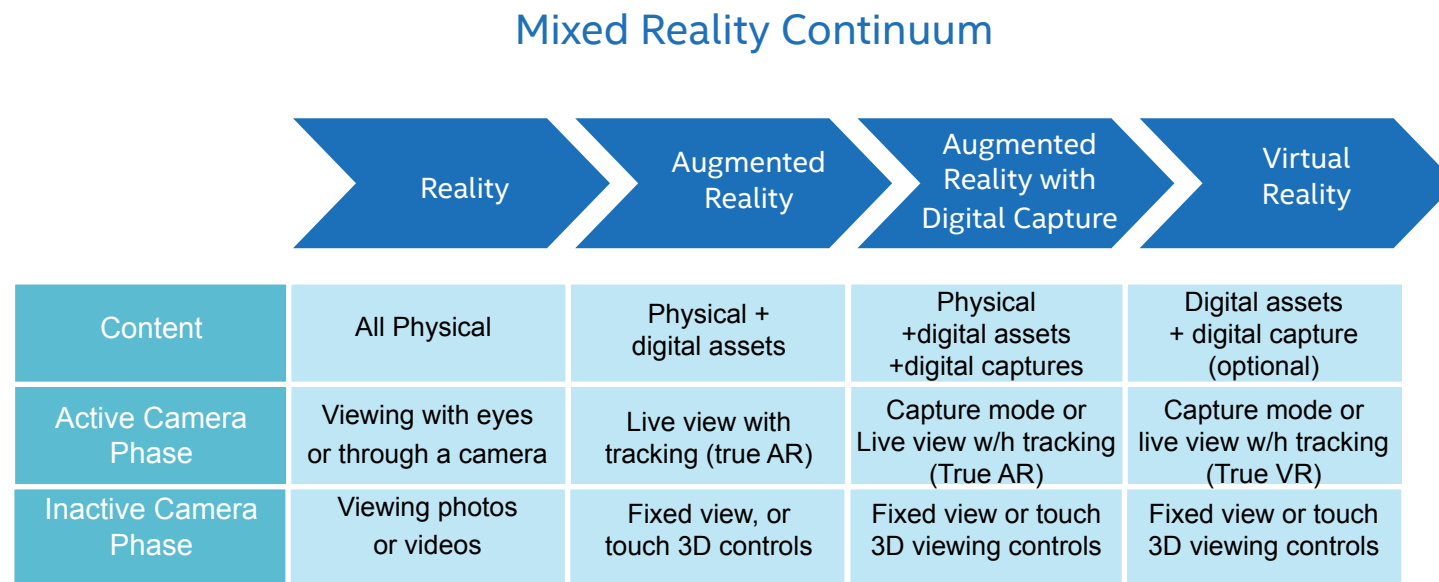
# Mixed and Augmented Reality

“Mixed Reality” refers to a range of experiences that blend digital and physical content. Augmented reality and virtual reality are the two most common of these.

In this section we provide recommendations for designing good mixed reality applications for tablets with rear-facing depth cameras. In addition we discuss the basics of camera tracking and localization, which is central to most mixed reality applications.

# Overview of Mixed and Augmented Reality

We use the term **Mixed Reality** to describe a range of experiences that blend physical (real) and digital (virtual) worlds. These can be mapped onto a scale according to the relative amount of physical and digital content — from everyday reality, which has all physical content, to virtual reality, which has all digital content.† The following chart shows three different mixed-reality experiences (Augmented Reality, Augmented Reality with Digital Capture, and Virtual Reality). The active-camera and inactive-camera rows describe the corresponding phases of these applications.



In the following sections we discuss:

- Guidelines on camera tracking and graphics that apply to all mixed reality applications.
- Guidelines for particular styles of mixed reality applications that can be built with the camera and SDK.

† Adapted from Paul Milgram and Fumio Kishino, "A Taxonomy of Mixed Reality Visual Displays." IEICE Transactions on Information Systems, Vol E77-D, No.12 December 1994.

# Overview of Camera Tracking and Localization

## Tracking

Tracking refers to the system's ability to determine the camera's position and orientation with respect to the world, and update this information in real-time as the user moves. Since the system knows the actual distance to objects in the scene it can triangulate from multiple points to track where the camera is over time.

The Intel RealSense SDK provides tracking through the [Scene Perception](#) module. The tracking is based on both the depth data and RGB data.

Tracking is a requirement for any kind of virtual reality or augmented reality experience that gives the user a dynamic view as they move around a virtual or mixed-reality scene. When implemented on a tablet, this idea is sometimes referred to as a “magic window” experience.

## Localization and Scene Perception

Localization is the term for the system's ability to remember the scene and restart tracking quickly when a new session starts.

To perform tracking, localization, and mesh reconstruction, the system forms a model of the surfaces in the world and updates it as the camera moves. The Intel RealSense SDK calls this computer vision method “Scene Perception.”

For scene perception to work well,

- The scene needs to have enough objects that the camera can see well. (See the section on difficult environments for more information.)
- Very fast motion by the user should be avoided.
- The tracking needs a few moments to establish a reasonable model and begin accurate tracking. This might mean your application needs an explicit “scan” phase to initialize.
- Guidelines related to these issues are discussed in the following pages.



# Camera Tracking: Setup and Preview

Tracking data will be reported immediately (and at 30+ frames per second afterward) when activated in the scene perception module. But it may require a few seconds of motion before the results become robust. Consider the following ways of dealing with this initial period of tracking:

- Filter or ignore early data (don't attach it to a live scene camera) and wait for tracking quality (as reported by the SDK) to increase.
- Encourage the user to begin with simple motions. Moving the tablet from side-to-side over a feature-rich scene will help to initialize the tracking.

**When tracking is not possible.** Tracking is essential for certain mixed-reality experiences. But keep in mind that most applications should have both “live” active-camera and “offline” inactive-camera modes. Allow the user to begin with inactive-camera mode when possible. (See later in this section for more guidelines regarding active-camera and inactive-camera modes for mixed reality.)

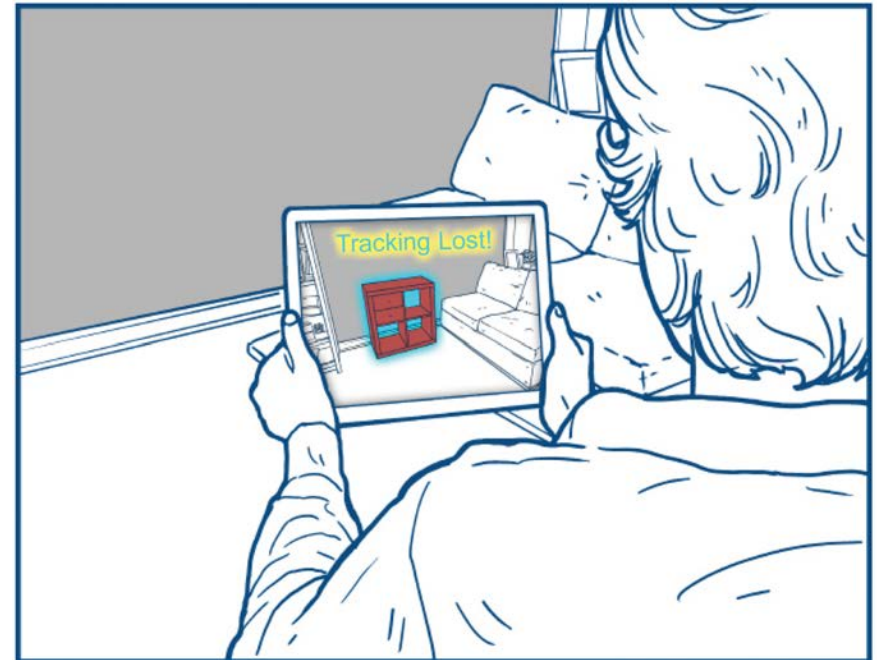


# Camera Tracking: Feedback and Re-Initialization

When tracking is working, the user will see the graphics update in real time as he or she moves the tablet. No additional feedback is needed when tracking is performing normally.

If the tracking quality degrades, the user will see the graphics become jittery, and the real and virtual elements in AR will start to lose registration. If the tracking module reports a low quality measure, use this to warn the user that tracking is poor. The easiest causes to identify are if the user is moving too quickly, or the user is moving out of the original scene. In those cases, instruct the user with clear text alerts to slow or to remain in range.

If tracking stops completely, the graphics will stop updating (or at least the virtual camera will stop moving), and the real and virtual elements in AR will lose registration. To re-initialize tracking, instruct the user to move back to their prior location (and re-align the scene), or start over (by pressing a reset button).



# Graphics Considerations for Mixed and Augmented Reality

When rendering mixed reality scenes, consider the following guidelines for making virtual content blend well with physical content and making alerts and annotations stand out clearly. Of course in some games you might aim for different effects and want your virtual objects to look cartoonish or artificial. The following suggestions are mainly about realistic applications, such as visualizing virtual furniture in a real environment.

**Lighting:** enable shadows and render your virtual scene with multiple area light sources in order to give a more natural look with softer shadows. Enable shadows. Test your application with users in real environments and adjust the brightness of your virtual content so that it blends well.

**Materials and objects:** use texture mapping and unsaturated colors for virtual objects to give them a more natural appearance. Avoid highly reflective or semi-transparent virtual objects.

**2D Elements and overlays.** Annotations, alerts, and touch controls should be legible against the active-camera view of real scenes. Avoid saturated colors. White or light colors for lines and text generally work best, and add contrasting outlines or shadows.

**Gravity and up-direction.** Use data from the device's Inertial Measurement Unit (IMU) to properly orient the virtual scene and assign a direction for gravity.

**Use animated transitions.** When adding a virtual object or character into a scene, use animations to draw the user's attention (for example a character dropping onto the terrain). Highlight selected virtual objects and animate editing actions.

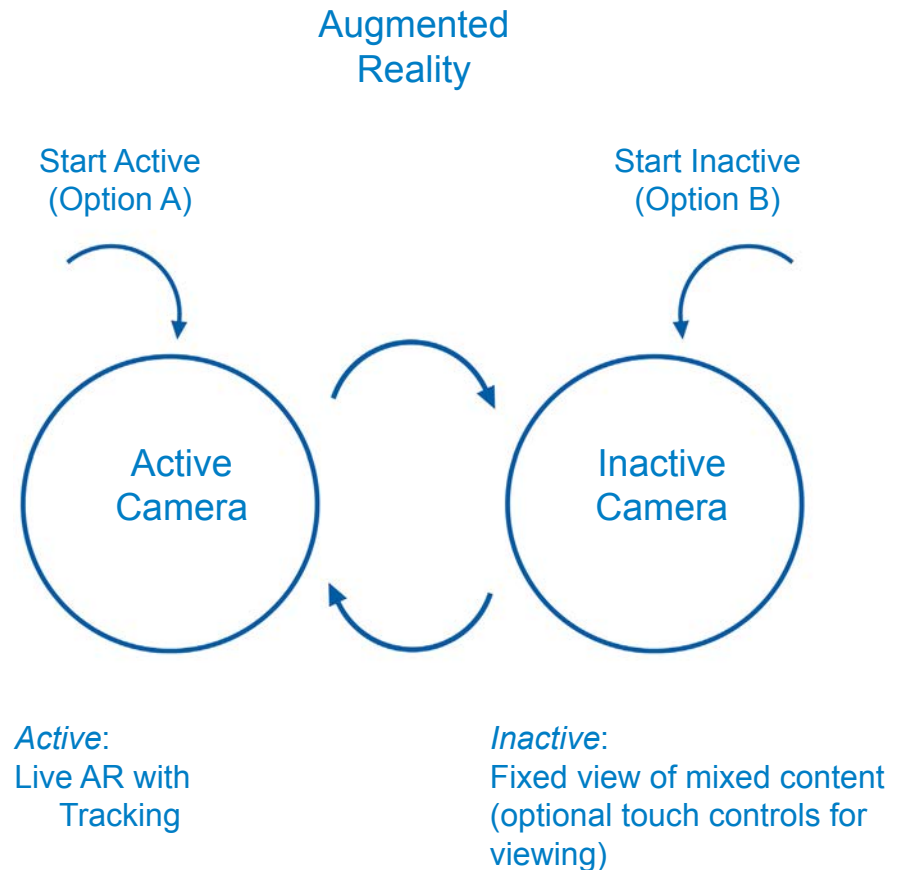
**Transform the real scene.** In mixed and augmented reality applications, your application can use the scene perception module to capture a mesh of the real scene. Consider applying procedural effects, textures, and other transformations to this mesh to create compelling effects.

# Augmented Reality

**Active Camera (True Augmented Reality).** Augmented reality (AR) is the idea of showing a live view of a real scene augmented with digital content registered in the same space. These augmentations can be information, such as repair instructions, or objects, such as virtual furniture. As the user moves the tablet they see a live, updated view of the physical and digital content, as though they are looking through a magic window.

**Inactive Camera.** In most applications, it's useful to offer an inactive-camera mode, when the view is fixed or adjustable with 3D viewing controls on the touchscreen. As we mentioned earlier, you should limit the duration of active-camera interactions so that users don't become tired or sore from holding the tablet. With an inactive-camera mode, users can rest the tablet and do more precise touchscreen interactions.

**Transitioning.** Your app can start in either active or inactive-camera mode, and flow between the two modes, as illustrated here. Most AR apps should start in active-camera mode to help orient the user.

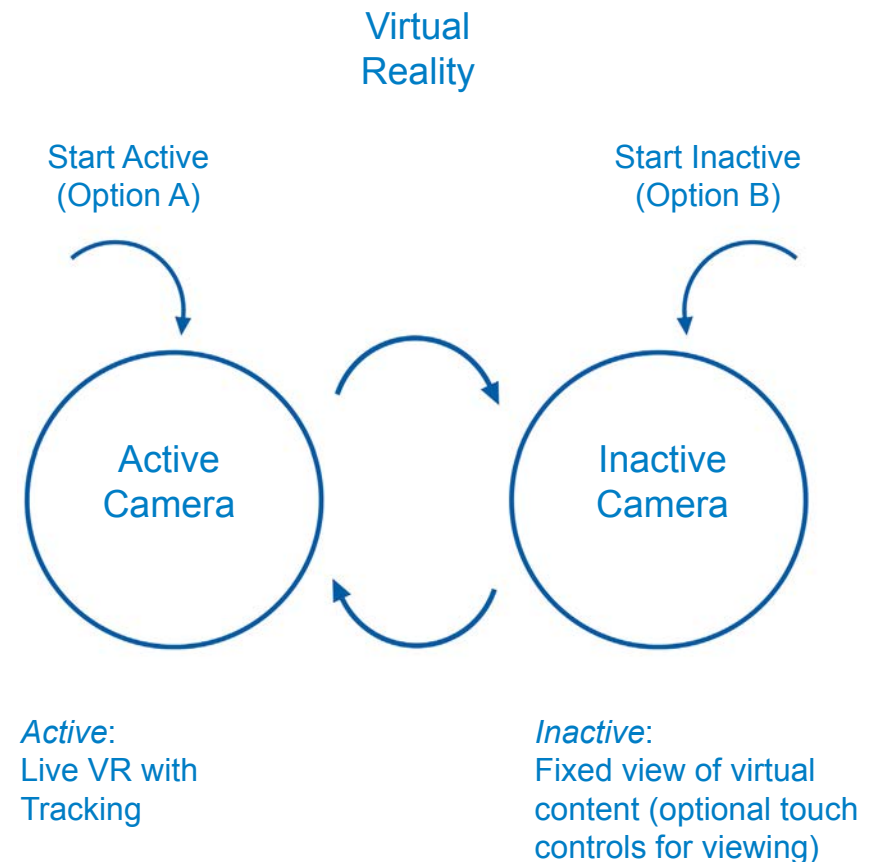


# Virtual Reality

**Active Camera (True Virtual Reality).** Virtual reality (VR) is the idea of viewing a digital scene with dynamic perspective that updates as the user moves. This is most commonly done using a head-mounted display and head-tracking. In our scenario, the display is on a tablet and tracking is provided by the Intel RealSense Camera. As the user moves the tablet, the view of the digital scene changes, as if the display is a magic window into another space (like a sci-fi parallel universe). The difference between VR and AR is that the content is fully digital in VR instead of a mix of digital and physical.

**Inactive Camera.** For the same reasons as for AR tablet apps, it's useful to offer an inactive-camera mode, when the view is either fixed or adjustable with 3D viewing controls on the touchscreen.

**Transitioning.** Your app can start in either active or inactive-camera mode, and flow between the two modes, as illustrated here. Starting in active-camera mode will help orient the user to the virtual space. But starting in inactive-camera mode could be useful if you first want the user to get a sense of what the digital world looks like, without the added burden of learning how to move around the space.





# AR and VR with Digital Capture

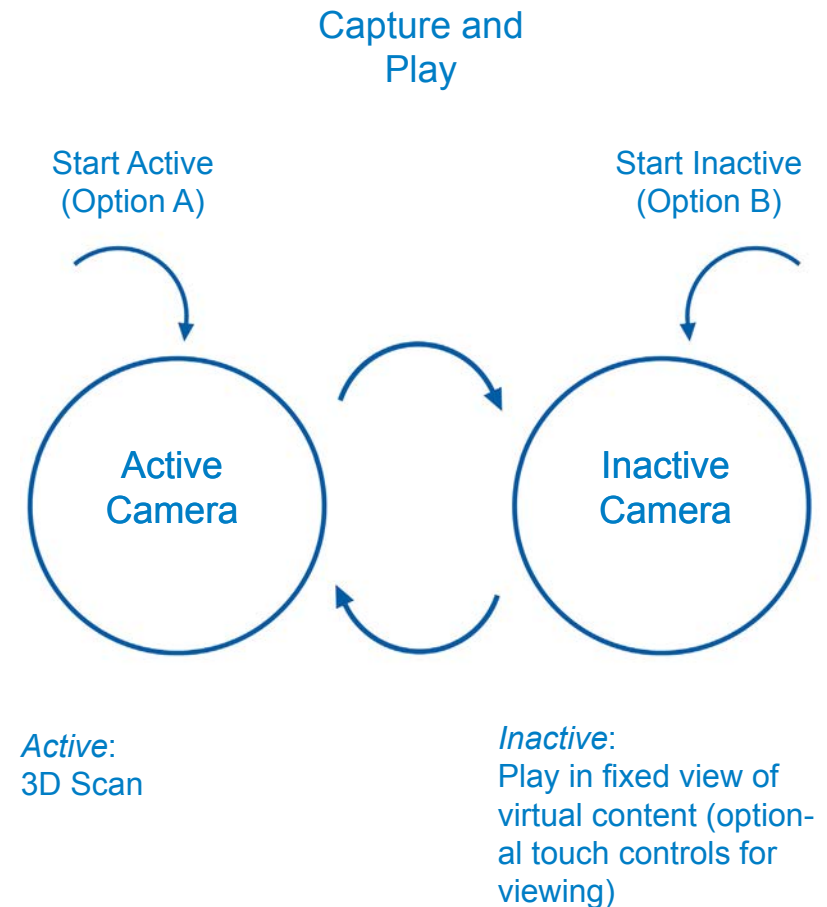
“Capture and Play” is a term we use to describe games that involve an element of 3D scanning. The user can perform 3D scanning on an object and then insert that model into the game. This could be done in either an augmented reality or virtual reality game, and the concept could be applied to other applications, not just games.

**Active Camera for 3D scanning.** Capturing objects will happen in active-camera mode, and the design for this will be similar to that for 3D scanning applications.

**Inactive Camera.** As with regular AR and VR tablet apps, it's useful to offer an inactive-camera mode, when the view is either fixed or adjustable with 3D viewing controls on the touchscreen. The only difference for an AR/VR with digital capture app is that the content will include digital objects that were previously captured.

**Active Camera for Live Viewing.** You can also provide a live, dynamic view of the digital scene. This is the same as the active-camera phase of AR and VR tablet apps as described previously.

**Transitioning.** A “capture and play” app should normally begin with an inactive-camera phase to orient the user. The active-camera phase for 3D scanning should begin and end using an explicit command.





# Games

This section covers several considerations, challenges, and principles common to building with the Intel® RealSense™ Camera (R200).

It includes general methods for describing the interaction space with the camera in a rear-facing configuration in the game context. It will offer some suggestions for approaching the challenges of game development in this interaction space. It will provide information concerning some known challenges and mitigation techniques for developers working in this space.

# Some Situations to Consider

The following pages present a number of design models for describing challenges, affordances, and possible solutions for various aspects of Mixed Reality Games.

They include methods to model the problems in descriptive ways, considerations of varying approaches, and suggestions for determining appropriate solutions.

As this space is largely unmapped and lacking in extensive exemplars of design at this time, the design models are few. However, they are informed by early experimentation, prototyping, and testing efforts.

This document aims to assist the developer in avoiding some of the pitfalls discovered in early efforts.

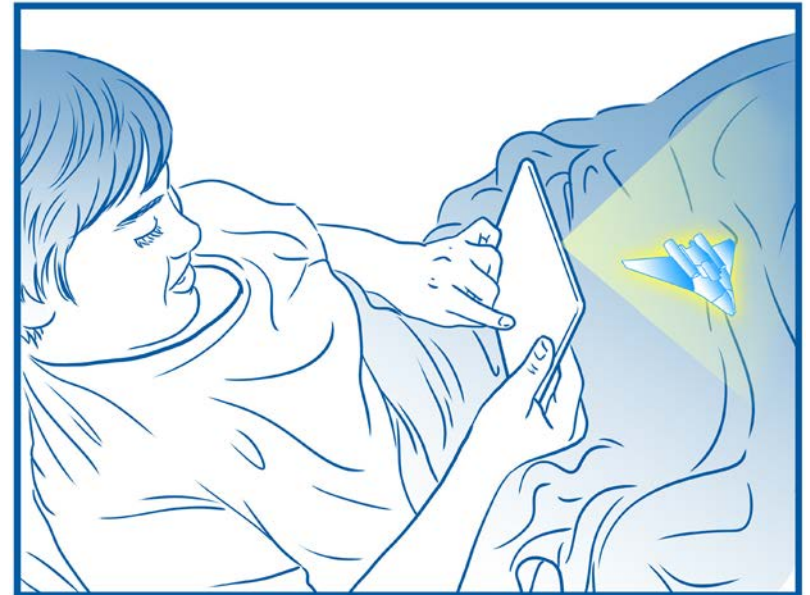


# General Considerations for Mixed Reality Games

Games built with the “R200” camera and Intel RealSense SDK present a range of possible game play scenarios. These can include mixed realities of varying flavors ranging from what has traditionally been called Augmented Reality toward immersive experiences previously described as Virtual Reality. There are many varieties of mixed realities that contain elements of each of these well documented types of experiences as well as new elements that lie outside either of these well known forms.

Mixed Reality Games might include characteristics such as:

- Physical environments becoming part of virtual game environments.
- Rendering virtual elements in a realistic manner on your physical environment.
- Changes in the physical world creating change in the virtual world.
- Virtual world elements interacting with and changing the virtual representation of the physical world.
- Altering the physical world in ways that render it fantastical when experienced within the game space.



# Challenges to Introducing New Interaction Modalities

Whenever developers seek to introduce new interaction modalities to the player's experience, the player will be challenged to meaningfully understand and use the system in a way that allows for intuitive action. Fortunately, these challenges can be addressed by using many of the methods that are already common in developing game interactions. One difference is that while many of the interaction systems for existing modalities are well known, those for new modalities must be built and trained. This means the developer must enable the player to learn and adapt both mentally and physically as they build new mental models and physiological motor programs.

## Players:

- Learn techniques to communicate their intent in specific ways comprehensible by the system.
- Learn to interpret feedback rapidly enough to adjust their performance of interaction techniques on the fly.
- Develop specific motor skills to execute those techniques with a significant degree of accuracy while adapting to feedback.
- Develop models of understanding for adjusting the performance of their actions, troubleshooting, diagnosing unexpected system behavior, and adjusting their behavior or system parameters to produce desired results.

## Developers:

- Develop specific interaction mechanics that can be interpreted by the system with a high degree of fault tolerance.
- Develop a coherent system of interaction mechanics that provide players a collection of techniques to communicate their intent to the system.
- Provide players with meaningful feedback that communicates their context within the system.
- Provide players with feedback concerning how the system is interpreting their current behavior in relation to the suite of interaction mechanics that can be triggered with player techniques.

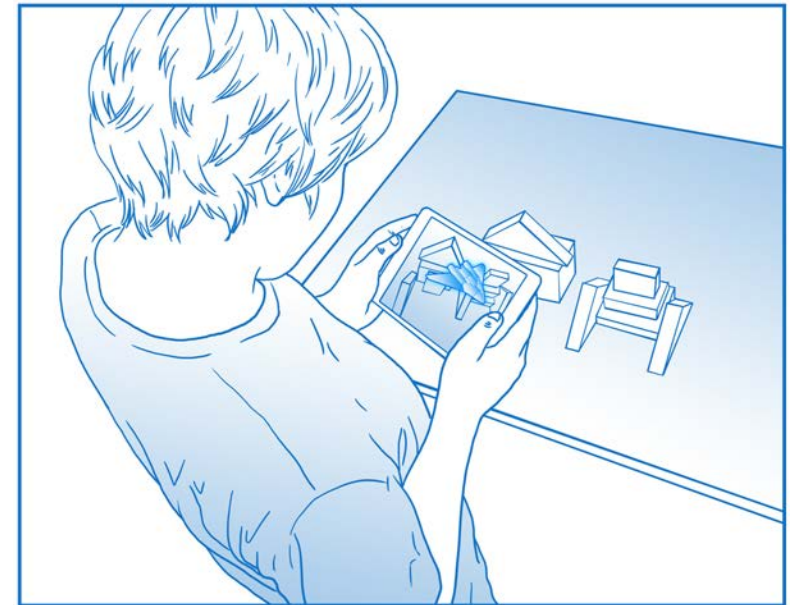


# Challenges of Designing for Mixed Reality Games

Mixed Reality Games present a variety of challenges dependent on what blend of virtual and physical elements the game developer chooses to utilize and how she seeks to integrate those into game play.

Much like the player's control of the camera in first and third person games presents specific challenges to staging dramatic scenes and effectively communicating timely information, Mixed Reality Games may sometimes give players significant control over level design itself.

- The playable game environment may be “unknown” by the designer during the making of the game.
- The developer should assume “anything” or “anywhere” when designing for play.
- The developer must consider the challenges of designing for a player that has control of level design and find ways to effectively design compelling play that takes advantage of this situation.
- The developer needs to find ways that create meaningful play for working in this blended world.



# Continuity of Experience

Games often involve a full world from a visual and story arc perspective. When building mixed realities, developers need to maintain a cohesive style and one where the elements of the blended world **make sense** and are **believable** within that game world. This can be maintained by crafting the scanning phase art assets and language to match the theme and style of the gameworld, and in some cases, by making the scanning phase relevant and integrated into the gameplay.

- Like other aspects of game design, interactions that blend physical and virtual elements contribute to the landscape of the player experience. These elements should feel as comfortable, cohesive, believable, and integrated into the gameplay experience as traditional elements of art, design, mechanics, systems, etc.
- There should be a harmony and rhythm of interactions that take into account the novelty of these experiences while not appearing forced or inconsistent with the traditional mechanic systems.
- Some of the technical challenges of tracking, scene perception, etc. must be considered when designing gameplay. For instance, the player cannot move too fast without losing tracking and requiring a recalibration. This can be discouraged by considering the **momentum of gameplay** leading into the capture phases of play.



# Reconstruction and Scene Perception as the Ground of Play

Two primary modalities of interacting with the physical world in Mixed Reality Games are Active Camera and Inactive Camera modes. In each mode, the camera can be used in gameplay in a number of fashions. Each mode presents certain kinds of opportunities for play, but also come with constraints as described below.

## Active Camera Play

- The camera is live, and the user is holding the tablet up to capture something or use tracking to control a virtual camera or perspective
- Provides physical tracking based camera controls used to manipulate a virtual camera in virtual game space
- Enables AR style game play that overlays virtual elements over the physical world in real-time
- Can significantly alter the virtual representation of the physical world in a magic window kind of interaction
- Uses significantly more power over time as the camera and IR emitter are powered on for a longer duration
- Requires that the player have enough environment available to move about and enable game play
- Must make performance trade-offs for various features due to the intensity of scene perception, procedural set-dressing, rendering, and game play.

## Inactive Camera Play

- The camera is off and the user is interacting with offline/captured content
- Scan and save physical environments in active camera mode as virtual game maps to be played anywhere using traditional mobile game design methods in inactive play mode
- Allows for procedurally set-dressing of captured scenes during a cool down phase after the physical scanning
- Somewhat alleviates the fatigue associated with holding a tablet upright for extended periods when looking at the world through it
- Breaks reconstruction and procedural set-dressing into distinctive phases from game play and rendering
- Requires a scanning phase to thoroughly capture all playable spaces of a game before game play

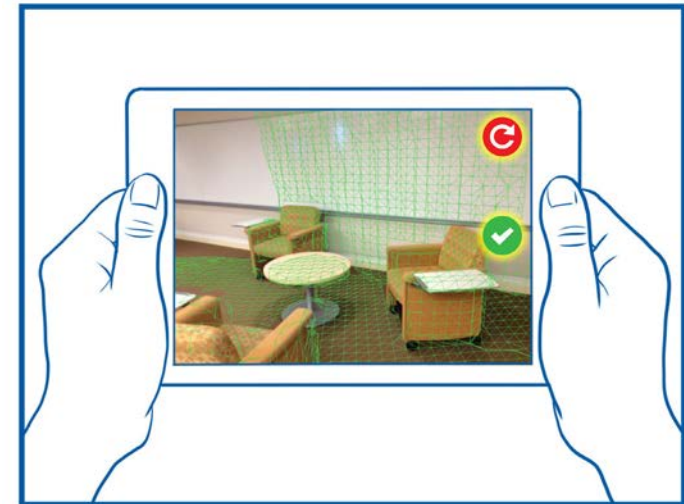
# Pre-Visualization of Capture Targets

In Capture and Play type scenarios, the player will be capturing an area or object and converting it into a virtual element or space. What will be captured should be communicated to the player in ways that allow the player to make meaningful decisions about what they are scanning and to make adjustments if they are not capturing their intended target.

Some areas to consider:

- How does the game visually communicate the capture area to the player?
- How does the game communicate what has been captured and what has not?
- How does the game communicate tracing status to the player?
  - Can the game help the player to recover before fully losing tracking?
  - Can the game help the player re-localize once tracking is lost?
- Do we want to communicate capture progress to the player?
  - How does the game do that?
- How does the game communicate successful/unsuccessful scans?
- How does the game communicate the capture scale relative to the player perspective?

For more information, refer to the 3D Scanning and Mixed Reality sections earlier.



# Physical Capture as Configuration or Gameplay

The demands of interaction design for the capture of a physical object or environment will vary whether the phase is being treated as a configuration phase of the game or whether it is being treated as part of gameplay. Configuration phases can be less intensive and immersive in terms of player experience and more forgiving in terms of time for more accurate rendering. In-game capture will require more believability with respect to the game world, but can be more forgiving in accuracy if the game world has a simplified style or mechanics are used to explain away inconsistency between physical and virtual. With either approach, the developer should provide appropriate feedback to the player as to the capture area, the status of the capture, and risk of losing tracking.

## Capture as Configuration

- High importance placed on accuracy of capture.
- Not driven in game-time and providing a buffer for capture and render processing times.
- Requires more thorough scanning process to assure high quality capture.
- Failure intolerant with respect to players failing at producing a high quality scan on their first attempt.
- Tolerant to buffer times between scan and rendering of the captured environment.

## Capture as Gameplay

- Believability of capture phase integrated with virtual aspects of gameplay including style, theme, and mechanic systems.
- Can be more fault tolerant if the world is simplified/stylized in ways that mask the inconsistencies with believable repairs and transformations in reliable ways.
- Requires instant feedback and performance to maintain the feel of having an in-game experience.
- Failure tolerant if carefully tied to skills development in a way that allows the player to make informed decisions and to adjust accordingly as they would other aspects of play.

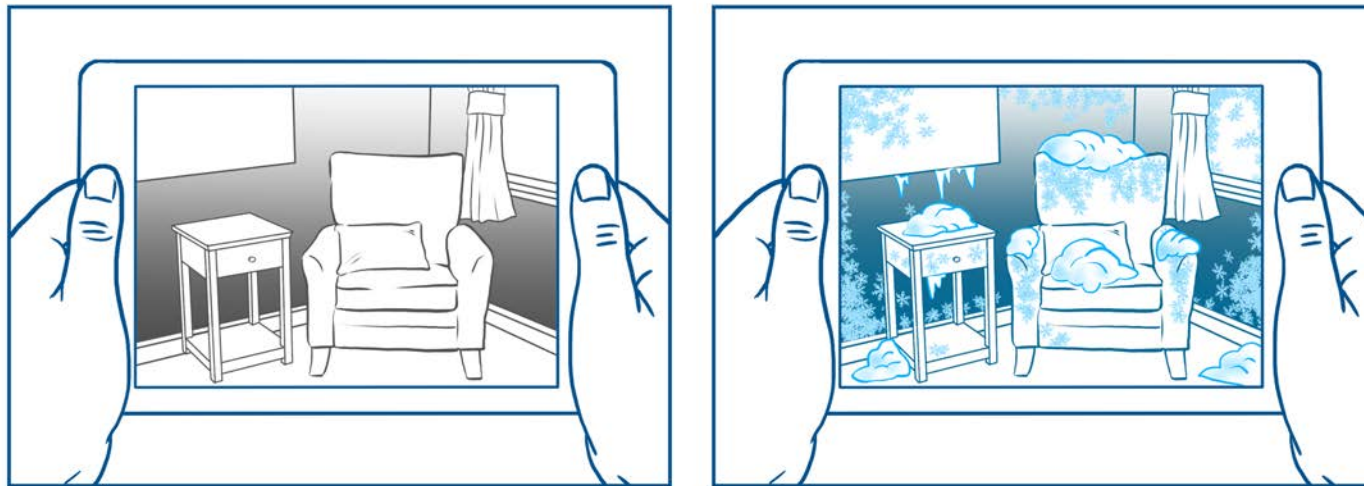


# Procedural Set-Dressing

Procedural set dressing is a powerful set of techniques for broadening the applicability of environmental capture to a wider reach of games. While some situations might call for accurate, naturalistic capture and rendering. Many situations may wish to transform the captured environment to reflect heavily stylized game worlds.

In these situations, we might use various forms of scene analysis to create behavioral rule sets for generating appropriate game content. These may vary in complexity from simple height, color, and size analysis, to more complex shape or object recognition. These analyses can feed a system that produces complex variation from a relatively small and simple rule set.

- Procedural modification of the physical terrain - e.g. blue tinge and add snow/ice for a frozen world; age objects in the real world so it looks like a worn version of today's world, alter the appearance to the point where only the shape of the space remains visually similar to the real space
- Conversion or modification of the physical world is an element of the story

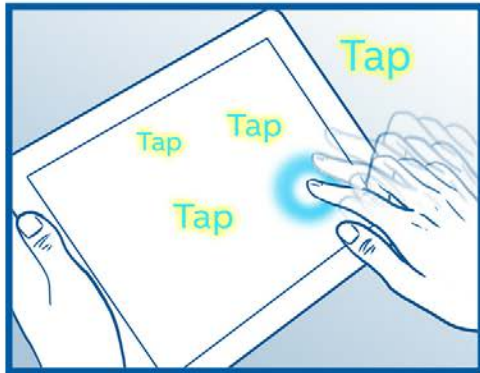


# Momentum of Play

- Slowing the player's movement and actions before any activity requiring tracking of the camera position can minimize the amount of loss and re-acquisition of position necessary from camera tracking loss.
- The player will experience **physiological affects** of movement, excitement, fatigue, etc. The current state of those affects will affect the experience and performance of any activities involving movement with the camera in active mode.
- An excited player may move less fluidly and more quickly. This can affect the quality of tracking in scene perception types of activities. If the **excited player** needs to perform a thorough scan or capture, they may move more erratically and with less accuracy with respect to their intended scan target.
- Consider using **cool-down buffers** between vigorous and intense gameplay and other sorts of activities requiring accurate movement of the camera in order to capture environmental information.



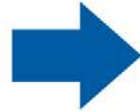
# Player Cool Down Buffer



If the game loop as the player actively playing in rushed, time-sensitive gameplay against numerous enemies



Inserting a cool down buffer, such as a round-up of the last rounds statistics or a communication intervention with another character talking in a calm and directive voice, etc.



Due to the momentum of gameplay and the associated physiological effects, the player is likely to move more quickly and with rapid acceleration resulting in lost tracking.



Can help reduce the physiological effects of excitement, and lead the player into a more relaxed mode of interaction, resulting in better tracking and a better overall experience.



# Active Camera Tracking as Game Controls

The Intel RealSense Camera can be used in ways other than capturing depth data in order to generate meshes, etc for play scenarios.

One such way is for active camera tracking RGB and depth streams in order to actively orient the camera within its physical space. This active camera tracking is vital to orientation when capturing data for scene perception based interactions. This capability can also be used as a form game control. Active Camera Tracking can be used in artist-created virtual game environments or in game environments generated using one or more of techniques associated with Capture and Play.

For instance, one might pair the camera tracking position in real world to a virtual camera within a virtual game environment. This would allow the player to look around a completely virtualized environment through a magic window type of interaction into a virtualized space.

This information can be used in relation with IMU, GPS, accelerometer, and gyroscopic data to generate a very rich understanding of player movement and viewing orientation.





# Augmented Reality vs. Capture and Play

In some gameplay scenarios, we will be playing the game in an **active camera mode** wherein we capture the environment as we play. This mode will render an environment created from scene perception middleware in real-time and alter it with virtual elements. We call this an **Augmented Reality** type of interaction.

In other situations, the player may have gameplay in which the active camera **capturing phase** of play is discrete from a phase of gameplay that takes place in a virtual space with all or parts of its content generated from the capture phase. This gameplay can include either **active** or **inactive camera modes**. We call this mode of discrete phases of environment capture and play in the generated virtual space, **capture and play**.

Augmented Reality  
(Active Camera)



Capture and Play



Active Camera



Inactive Camera



# A Continuum of Mixed Reality Games

Artist-Generated Virtual World with Active Camera Controls	Capture and Play with Traditional Controls	Capture and Play with Active Camera Controls	Augmented Reality
<ul style="list-style-type: none"> <li>• Traditional, artist generated virtual world</li> <li>• Active camera game control (such as virtual camera control)</li> </ul>	<ul style="list-style-type: none"> <li>• Active camera capture of environment</li> <li>• Asynchronous render phase of the play space</li> <li>• Possible virtual augmentation such as procedural set-dressing</li> <li>• Traditional game controls</li> </ul>	<ul style="list-style-type: none"> <li>• Active camera capture of environment</li> <li>• Asynchronous render phase</li> <li>• Possible virtual augmentation such as procedural set-dressing</li> <li>• Active camera game control (such as virtual camera control) for some parts of game</li> </ul>	<ul style="list-style-type: none"> <li>• Active camera capture</li> <li>• Synchronous camera capture and render</li> <li>• Active camera control</li> </ul>

# Artist Generated Virtual World with Active Camera Controls

A game utilizing an artist-generated virtual world with active camera controls is perhaps the least complex use of the RealSense™ camera technology in terms of technical implementation, but provides it's own challenges in design.

- One example might be a game in which there are pre-crafted puzzles with artist-generated assets as would normally be the case in game development. If the game was one that was based on visual perspective to solve puzzles, the player might use active camera controls to move around and look from various perspectives in the virtual world in a sort of magic window experience to solve such perspective puzzles.
- Another example might be a space shooter in which the player can rotate and look about the world as space ships fly by attacking. The player could rotate and fire using tablet controls at the receding ships that have passed.

Each of the cases cited above raises specific issues, such as:

- fatigue caused by moving the tablet around at arms' length
- risk of dropping or banging the tablet against objects in the player's physical environment

Mitigation techniques might include:

- Pacing the game so that the active camera control portions are short and separated by periods of other gameplay for rest
- Encouraging/rewarding the player to remain seated or in a fixed position when looking about the environment

# Capture and Play with Traditional Controls

A game that uses Capture and Play to capture an environment and to generate an environment that is then playable with traditional game controls presents unique challenges to game design. The designer must find ways of either encouraging the player to make level design choices that result in interesting environments that are playable in compelling ways, to make adjustments to the level that render it playable and fun, or a combination of both of these.

- One example would be a game in which the player knows the kinds of enemies, resources, and tools available to her. The physical level layout becomes a sort of challenge to the player to craft the game space that grants them the best opportunities to gather desired resources while providing obstacles for enemies that take advantage of the tools available to the player. This affords a sort of art-of-war gameplay in which the player must determine the best environment to tackle a specific challenge provided by enemies and available tools to combat them.
- Another example might be a story-driven game wherein the scanned environment is set-dressed to match the game world theme and style. Due to the unpredictability of the environments produced by the player, the developer may choose to add gateways within the player generated level to artist-generated level spaces that allow for specific story moments or events. For this type of game, the capture phase should make sense as a phase of gameplay in order to not feel arbitrary or discontinuous with the rest of the game.

In any such game, the most significant challenges to developers are the ability for the player to navigate the level effectively and for the gameplay to be compelling in the absence of traditional level design techniques. The following pages will elaborate upon these challenges and mitigation techniques.

# Player Movement Challenges

When using scene perception to generate playable environments in either Augmented Reality or Capture and Play modalities, there will arise challenges to environments not explicitly designed for a game character's or vehicle's movement. Some of these might include:

- Lack of navigable surfaces between high and low zones of the environment.
- Holes caused by missing information due to lack of scanning, black or reflective objects, etc.

There are a number of ways to approach these challenges, such as:

- Accepting them as constraints and designing gameplay that takes advantage of the situation as much as possible.
- Providing the player with mechanics that overcome specific challenges such as separation between low and high areas (climbing, jet packs, etc.)
- Building in environmental mechanics to assist through scene analysis (teleporters, launch pads, etc.)
- Using procedural set-dressing to mitigate these challenges (hole-filling, bridge creation, hanging ropes from high places, etc.)

Regardless of the challenges and the solutions, the developer should keep in mind that these practical solutions remain coherent with the continuity of the experience and the unity of the design.

# Example Challenge and Mitigation Techniques

## Challenge:

Player may scan an environment with height variance greater than player's ability to navigate.



### Player Mechanics

Provide player with mechanics that mitigate such issues, such as jump boots, rocket packs, etc.

### Procedural Set-Dressing

Procedurally set-dress environment by analyzing scene and inserting navigation assistance, such as ladders, teleporters, jump pads, etc.

### Scan-Phase Feedback and Coaching

Provide real time analysis during scanning phase of gameplay for known issues (such as height variation) and provide suggestions to player for avoid or correct

### System Design and Skill Trees

Make level design part of gameplay wherein players are coached on such challenges by mentors, and they must repeatedly recreate levels in order to develop their level design and accomplish missions.



# Capture and Play with Active Camera Controls

A game utilizing capture and play types of gameplay along with active camera controls will grant the player control of both level design and camera. This presents complex and difficult design challenges to the developer along with the non-trivial technical implementations. These can be difficult games to realize, but may promise a continuity of experience for the player's control and interaction experience that could yield significant payoffs in terms of novel experience.

- One example of such a game might be to design tracks for single or multiplayer races or demolition derbies. The track layout phase would utilize capture and play features and the active camera controls might be used to look around the environment while using the accelerometer for steering.
- Another example might break the use of the camera further into distinct modes. The environment could be a storytelling world that is procedurally set-dressed and crafted with gateways to virtual spaces as mentioned in the previous Capture and Play with Traditional Controls examples. Once entering the virtual spaces, one might encounter artist-generated levels with carefully designed puzzles that use the active camera controls as cited in the Active Camera as Game Controls examples.

This type of play presents the same challenges as cited in the pages on Capture and Play with Traditional Controls and Active Camera as Game Controls. This particular combination complicates those challenges however, in that each set of gameplay has the potential of complicating or constraining the demands of the other.

# Augmented Reality

Augmented Reality games are easily the most well explored type of game described in this document. Even so, easy access by the public to hardware and software that renders AR type experiences reliable enough to generate compelling gameplay have been in short supply until very recently.

AR gameplay presents unique challenges on this list due to the necessity of playing while capture occurs. Due to the demands of analysis and rendering, the developer has little time and resources to adapt to undesired environmental qualities upon the gameplay.

Due to these conditions, the developer of an AR game should think of how to use the environment as it is in its being unknown to make for compelling gameplay.

- For instance, if the player is controlling the character, perhaps that player has access to the ability to transform, use a mech-suit with the ability to fly, or has access to rolling vehicles as well as flying vehicles at his disposal.
- Another example would be simply providing the ability to use grappling hooks and other such player mechanics to discover ways to navigate the space.

In such gameplay, the developer should plan carefully for the type of experience that they desire the player to have, to recognize that the environment will be unknown and unknowable, and to find a way of setting goals for the player that can be achieved regardless of the terrain qualities with the capabilities given to the player.

In addition, the developer should provide the player with opportunities to rest their arms from holding and interacting with the tablet after prolonged play sessions, as fatigue occurs rapidly.

# Capture Fidelity and World Building

The target audience and experience deeply affects choices in every aspect of game development including systems design, art style, mechanic complexity, performance accuracy requirements, etc. A game for a 5 year old with less developed motor skills will be significantly different in many aspects from a competitive first-person shooter or fast action real time strategy game. All these design, art, and interaction elements produce a cohesive **unity of design** for the game.

This should also inform design choices made concerning interactions and capturing mechanics for games made with Intel RealSense devices. The fidelity of the interaction accuracy, the expectations of control and performance capacity, and the resolution of captured environments can all be adjusted in design to provide the highest quality experience that best matches the fidelity and requirements of the game world.

## Examples

- If a game plans to significantly alter the appearance of the captured scene via **procedural set-dressing**, some aspects of the resolution of the capture ( such as the RGB channel ) might be diminished to free processing of resources for the set-dressing aspects of the activity.
- If capturing an object that will be miniaturized in the game to a relative size, the size of area being tracked or generated can be reduced significantly. This mesh will still be very high fidelity relative to game needs, and some **procedural mesh reduction** might be in order.
- If a game capture will require almost naturalistic rendering and high fidelity interactions between the player and a captured environment that renders the capture as it appears in the world, then highest fidelity capture settings might be necessary to remain believable to the player.
- If a game only requires the tracking aspects of scene perception without the environment capture ( to control a virtual camera in game space, for instance ), then using resource intensive mesh generation is an inefficient use of system resources for achieving the rapid framerates that such activities might require.

# Scale: Capture and the Game World

When capturing objects or scenes for use in virtual game worlds, the developer must always be aware of the challenges of scale. Knowing how size in the physical world will translate into size in game units is a choice that will have significant effects upon the rest of the development process.

- The [minimum distance](#) with which the camera can capture depth data will physically limit the resolution of what can be captured. Details that are too small to be read accurately within the operative range of the camera will be lost to the captured objects.
- The ratio of physical scan size to voxels generated will limit the detail of the object or environment captured with scene perception middleware. A large environment with a small number of voxels will require a great deal of solving and will generate a significantly smoothed product mesh.
- Providing the player some form of known reference for the size of a capture relative to the game world will provide them the ability to make meaningful choices, learn, and develop skills.
- Variance in scale from capture size to game world can provide opportunities to see our physical world in new and fantastic ways, but we must design for that while taking account of the player's ability to construct environments that will yield interesting and playable spaces within performance constraints.
- Variance in scale can present challenges to pathfinding for AI if not mitigated in some way as discussed in [Sample Challenges and Mitigation Techniques](#).

# Multiplayer Synchronized Views

Multiplayer scenarios using Intel RealSense technology can take many forms.

- In the most basic, a single player can capture an environment in a **capture and play** modality. This environment may or may not be transformed significantly and saved as a game level that can now be shared and played just as any other game level. Multiple players might be using their tablet controls in a traditional control scenario, and only the capture phase really reflects new gameplay elements.
- In a second usage of the camera, each player might be looking into the virtual level created, using the scene perception tracking to control a virtual camera looking into a virtual world. This might be useful for strategy or building games.
- In a more complex scenario, each player might need to interact with the environment in an **active camera, synchronous** form of gameplay in which scene perception is generating the game environment on-the-fly while players have independent views of that environment. This would require significant design and development to maintain a **coherent and synchronous** game environment amongst all players as they move about interacting with the world as it is generated in real-time.





# Storytelling Without Level Design

One of the most significant sets of challenges presented to game developers when allowing players to generate level content is the lack of control that provides for sophisticated gameplay pacing tied to storytelling. This is and will be a significant challenge for some time to come.

It is reminiscent in many ways of the lack of control that computer games presented very early in relation to other visual storytelling forms such as film. This presented further challenges when developers provided **constrained control of the camera** to players within virtual 3D environments. Two decades later, a significant set of principles, techniques, and methods have been developed that provide developers with a powerful tool chest to assist a well-developed craft for storytelling in interactive 3D virtual environments.

Intel RealSense technology enables developers to provide players with the tools of level design as part of gameplay. How to do this effectively will require experimentation, iteration, and discovery. That being said, we have already begun to develop and discover some initial tools in the development of this new area in the craft of storytelling game development.

- **Making level creation a part of gameplay** can enable opportunities wherein the player is responsible for the levels in which her play takes place. Failure and improvement become part of the skills development cycle of the game.
- **Procedural set-dressing** can enable embedding of virtually constructed narrative elements within a level largely created by the player, but shaped by the developer. This is not entirely dissimilar from complex camera systems in third person adventure games.
- Tying player skill and asset unlocking to challenges presented by unfortunate level design choices can lead to new forms of compelling gameplay and storytelling wherein the player's character-based development and challenges are related to their **skill development with level creation**.
- As a skill development tree that makes sense in the game world– can assist players in becoming more effective level designers and enjoying more compelling gameplay.

# Encouraging Player Choices in Level Design

Some ways in which the developer might encourage good level design choices by the player when configuring and capturing their environments and levels are as follows:

- Provide enemies with specific range weapon abilities that make it nearly impossible for the player to combat them on an open field without variance in height and providing cover.
- Grant the player an ability or tool that enables them to navigate to spaces unreachable by opponents of certain types or levels.
- Make environmental/physics challenges that require specific height variance or slope lengths for the player to solve the puzzles.
- Provide specific qualities to vehicles so that proper design of a track can leverage benefits of specific vehicle qualities (e.g. better handling or rapid acceleration)
- Provide known patterns for procedural set-dressing that grant the player the ability to cultivate advantageous starting points for SIM building spaces, etc. (e.g. water availability, forests, mountains for ore, etc.



# Measurement

The Intel® RealSense™ Camera (R200) enables applications for measuring distances in photos or scenes. This section describes best practices for designing these experiences with the SDK.

# Measurement in Enhanced Photos

Measurement controls are similar to other editing controls for enhanced photo and video applications. (See guidelines earlier in this document.) The basic idea is that users can select points in the photo and get a measurement of the distance between them. Consider the following guidelines for implementing this experience-

- Support measurement actions during an inactive-camera mode. The user should take the photo then get measurements afterward by touching locations on the screen, as this will be more comfortable and precise.
- Assist the user in touching important points by snapping to object or layer boundaries when possible. Selecting a point on a touchscreen is difficult to do precisely; intelligent snapping will help with this as will magnifier and offset-style interactions.<sup>†</sup>
- Let marker points be adjusted. When the user touches a point, draw an adjustable marker at that location that the user can drag around the screen or delete.
- Preview the measurement result live as the user drags markers around the screen.
- Consider the eventual goal of measurement. Typically users will want to create a schematic or marked-up photo with several measures marked on it. Design for this by retaining measurements as they are done and providing effective mark-up and drawing tools. Allow the measurements to be easily exported and shared with others, for example, as an image with annotations. Allow the annotations to be hidden temporarily.
- Be aware of the accuracy limits to the camera and alert the user when the depth quality is low where measurements are being taken. Display measurements with an appropriate numbers of digits so as not to imply more accuracy. Don't implement measurement for scenarios where higher accuracy is required.

<sup>†</sup> see, for example: Vogel and Baudisch. "Shift: A Technique for Operating Pen-Based Interfaces Using Touch." In proceedings of CHI 2007.

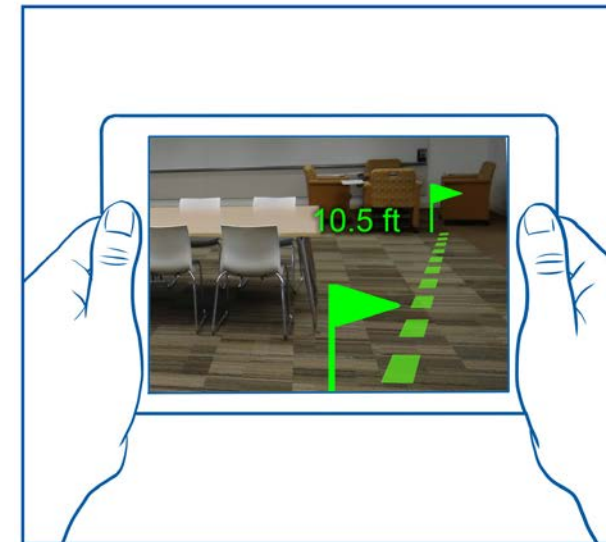
# Measurement in Scene Perception

The Scene Perception module allows you to record measurements as you move the tablet around in space while tracking is active (this is an **active camera mode interaction**). The concept here is that you can place two markers in the scene and find out the distance between them. What's different is that the user physically walks between the two positions in order to mark them.

Some things to keep in mind in this scenario:

As in the offline measurement case, allow the markers to be viewed and adjusted after the fact. You should also allow users to view the scene with an inactive-camera mode so that they can confirm the measurements more carefully.

Minimize the need to mark the same points multiple times. Instead of measuring distances one at a time with pairs of points, implement a scheme where the user can drop markers at key positions and then see all the distances together and make the connections as an inactive-camera mode interaction.







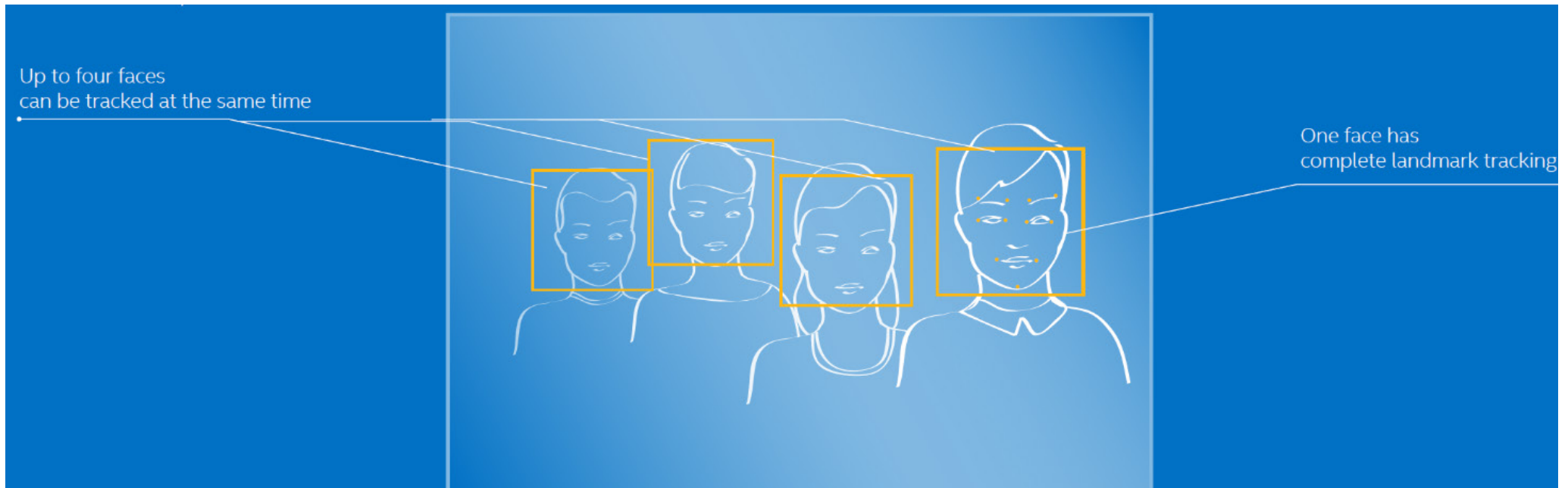
# Face Detection and Tracking

This section describes how you can use advanced face tracking with the Intel® RealSense™ SDK. The “R200” camera supports the same Face Module features that work with the “F200” camera, but keep in mind that the distance to your subject will typically be greater.

For additional guidelines on using the Face Module refer to the Intel® RealSense™ SDK Design Guidelines (F200).

# Face Detection

The Intel RealSense SDK provides accurate 3D detection and tracking of 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). You can get alerts for when the face is in the FOV, partially in the FOV, or occluded.



# Avatar Control

The SDK provides simple avatar control for use in our 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:

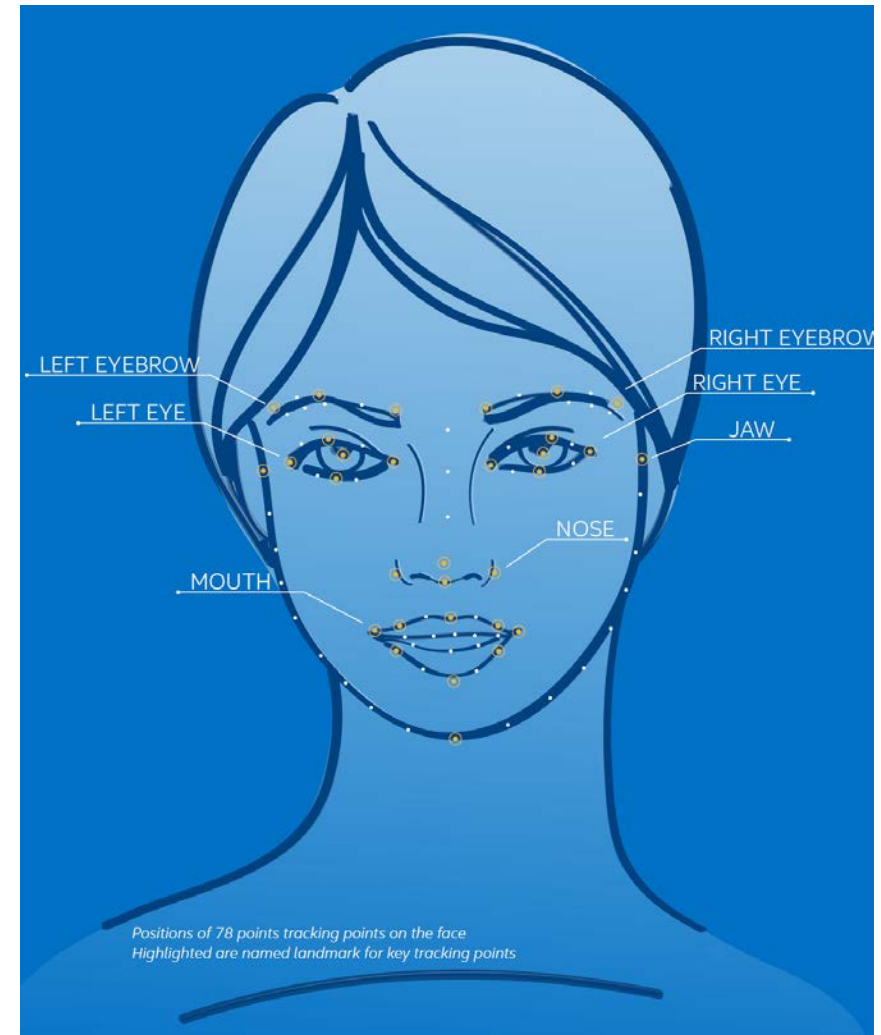
- Add your own avatar models
- Adjust smoothing on the landmarks and expressions for various usages and behaviors
- Enable/disable specific expressions or landmark subsets (e.g., you may want to move only the eyes on your avatar)
- Allow mirroring (e.g., when the user blinks their right eye, the left 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)

# Landmark Tracking

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 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.

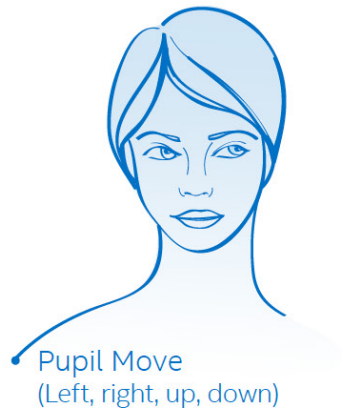
Landmark tracking is supported for users with and without facial hair and glasses. The system works best when faces are facing directly toward the camera (within about 15 degrees rotation in yaw, about the vertical axis).



# 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:







# Speech

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

The “R200” camera supports the same Speech Module features that work with the “F200” camera. For additional guidelines on using the Speech Module refer to the Intel® RealSense™ SDK Design Guidelines (F200).

# Best Practices for Speech Input

For the Intel® RealSense™ Camera (R200), speech is best used as an optional way to enter commands and modes. This can be helpful for enhanced photo and video applications, for example.

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).

## 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. 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 virtual button for “push to talk” control.

## Feedback

- Teach users how to use your system as they use it.
- 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 users know what commands are possible.
- Let users know that their commands have been understood. One easy way to do this is to relay back a command.
- If you give verbal feedback, make sure it is necessary, important, and concise! Don't overuse verbal feedback.