

Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Google ARCore

Marco Vettore
Matricola 1216433

Mattia Tamiazzo
Matricola 1219603

Alberto Varini
Matricola xxxxxx

Indice

1	Introduzione	1
2	Realtà aumentata	2
2.1	Storia	2
2.2	Applicazioni	3
3	Funzioni fondamentali	4
3.1	Motion tracking	4
3.2	Comprensione Ambientale	4
3.3	Light estimation	5
4	Depth understanding	9
4.1	Sessione ARCore con depth API	9
4.2	Depth Hit Test	10
5	User Interaction	12
6	Oriented Points	13
7	HitTest	14
8	Anchor and Trackable	17
9	Augmented Images	21
9.1	Capacità	21
9.2	Requisiti	22
9.3	Utilizzo della CPU e considerazioni sulle performance	22
9.4	Creazione del database	22
9.5	Come abilitare il tracciamento delle immagini	23
10	Instant Placement	24
10.1	Abilitare Instant Placement	24

	Titolo
11 Recording Playback	25
11.1 Compatibilità	25
11.2 Come vengono registrati i dati video e AR per la riproduzione	25
11.3 Ulteriori dati salvati	26
11.4 Come registrare	26
12 Augmented faces	27
12.1 Configurazione e utilizzo	27
13 Cloud anchors	30
13.1 Funzionamento	30
13.2 Implementazione e utilizzo	32
14 Geospatial API	33
Bibliografia	34

1. Introduzione

ARCore è un kit di sviluppo lanciato da Google nel mese di marzo 2018, utilizzabile nella maggior parte degli smartphone con Android Nougat o superiore (API level 24+). Tramite esso è possibile sviluppare applicazioni con funzionalità in realtà aumentata, permettendo all'utente di interagire con l'ambiente che lo circonda. In questo documento, dopo una breve panoramica sulla realtà aumentata, verranno analizzate le principali funzionalità e caratteristiche dell'SDK.

Nei primi capitoli vengono presentate le tre funzioni fondamentali del framework ARCore, che permettono al dispositivo di integrare contenuti virtuali al mondo reale:

- Il rilevamento del movimento, che consente ad esso di tracciare la propria posizione nel mondo.
- La comprensione ambientale, che permette la rilevazione della posizione e della dimensione delle superfici.
- La stima della luce, che consente di valutare le condizioni di illuminazione dell'ambiente.

Nei capitoli successivi sono invece presentate le funzionalità aggiuntive del framework, che consentono di migliorare l'integrazione tra virtuale e reale, come ad esempio il rilevamento della profondità, il posizionamento istantaneo, le API *Augmented Images* e *Augmented Faces* e altre importanti funzioni aggiuntive.

2. Realtà aumentata

La Realtà Aumentata (Augmented Reality - *AR*) è una tecnologia che permette di compiere esperienze interattive, in cui l'ambiente reale viene arricchito da contenuti virtuali. Similmente alla realtà virtuale, vengono creati elementi grafici sintetici con cui l'utente può interagire attraverso i sensi. Tuttavia, come spiegato in [2], nell'*AR* l'ambiente reale gioca un ruolo fondamentale: lo scopo della realtà aumentata è proprio cercare di collegare il mondo reale con quello virtuale.

L'*AR* viene definita in [1] come un sistema che incorpora tre caratteristiche principali: la combinazione tra reale e virtuale, l'interazione real-time e la rappresentazione 3D.

2.1 Storia

Il primo rudimentale sistema di realtà aumentata è stato creato da Ivan Sutherland [15] nel 1968. Esso era composto da un display ottico trasparente che veniva montato sulla testa e che poteva mostrare semplici immagini in tempo reale. Nel 1993 George Fitzmaurice ha creato *Chameleon* [9], un dispositivo che tramite un piccolo schermo collegato a una videocamera poteva essere orientato per esplorare uno spazio virtuale 3D. Simile al prototipo di Fitzmaurice, nel 1995 Jun Rekimoto e Katashi Nagao creano *NaviCam* [13], che prendendo in input un flusso video poteva riconoscere in real-time dei marcatori colorati e sovrapporre al video delle informazioni testuali.

Dal 2000 vengono creati altri primi sistemi di realtà aumentata, con applicazioni soprattutto a giochi interattivi, come ad esempio l'estensione *ARQuake* [16] o *Human Pacman* [4], ancora vincolati alle scarse prestazioni dei dispositivi mobili. Solo con l'aggiunta ai cellulari della fotocamera, e poi di schermi touch, vengono quindi create le prime applicazioni commerciali in grado di sfruttare le potenzialità della realtà aumentata. Ne sono esempi *AR Tennis* [10], primo gioco in AR collaborativo per cellulare, e *ARhrrrr!*, primo gioco mobile in realtà aumentata con contenuti grafici di alta qualità. Vengono quindi sviluppate le prime librerie software per la realtà aumentata, come *ARToolKit*, implementata prima in linguaggio C e poi in C++ nelle versioni più recenti, *OpenCV*, che possiede anche funzionalità per l'*AR*, e dal 2018 la libreria per Android *ARCore* di Google.

2.2 Applicazioni

Le applicazioni della realtà aumentata possono riguardare diversi ambiti, ai quali questa tecnologia può apportare benefici economici o qualitativi, oppure creare servizi innovativi [3].

In particolare, nel corso degli anni la tecnologia AR è stata usata per scopi pubblicitari e commerciali, quali la prova di capi d'abbigliamento senza doverli indossare o l'integrazione al marketing cartaceo di video promozionali tramite riconoscimento delle immagini, o per l'intrattenimento, come nello sviluppo di videogiochi. Trova inoltre applicazioni nella produzione industriale, in cui vengono sovrapposte all'area di lavoro istruzioni virtuali, nell'ambito militare, come l'addestramento al volo dei piloti, o per la formazione e la pratica sanitaria.

3. Funzioni fondamentali

TODo creazione di sessione? Nel seguente capitolo vengono

3.1 Motion tracking

ARCore usa un processo chiamato *Simultaneous localization and mapping (SLAM)* per determinare lo stato del dispositivo che si trova all'interno di un ambiente sconosciuto. Questo stato è descritto dalla sua posa (posizione e orientazione) che viene stimata attraverso prestazioni di odometria eccezionali e rilevazione di punti caratteristici. Con odometria si intende l'uso di dati ricavati da sensori di movimento che permettono di valutare il cambiamento della posizione nel tempo. Nel caso degli smartphone viene utilizzato il sensore IMU che rileva misure inerziali come la velocità, accelerazione e posizione. La rilevazione di punti caratteristici è l'individuazione di immagini con caratteristiche differenti che consentono al dispositivo di calcolare la sua posizione relativa. Questi punti di riferimento insieme alle misurazioni ricavate dai sensori permettono di avere una buona stima della posa e di ricavare la rappresentazione di una mappa dell'ambiente circostante. Tuttavia, il movimento sequenziale stimato dallo SLAM include un certo margine di errore che si accumula nel tempo causando una notevole deviazione dai valori reali. Una soluzione che può essere adottata per risolvere questo problema consiste nel considerare come punto di riferimento un luogo visitato in precedenza di cui si sono memorizzate le sue caratteristiche. Grazie alle informazioni di questo luogo è possibile minimizzare l'errore nella stima della posa.

I contenuti virtuali possono essere renderizzati nella giusta prospettiva allineando la posa della telecamera virtuale con quella calcolata da ARCore. Il contenuto virtuale sembra reale perché è sovrapposto all'immagine ottenuta dalla fotocamera del dispositivo.

3.2 Comprensione Ambientale

ARCore ha a disposizione funzione di comprensione dell'ambiente che gli consente di capire cosa circonda il dispositivo. Questa funzione è basata sulla ricerca di piani

Fonte:
<https://developers.google.com>

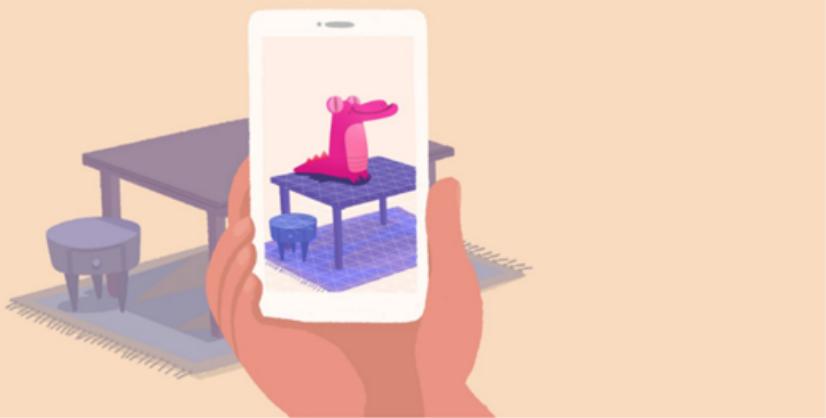


Figura 3.1: Esempio degli effetti prodotti dagli oggetti quando sono illuminati.

e vari punti caratteristici dell'ambiente.

ARCore si concentra nel trovare dei punti caratteristici posati lungo piani verticali e orizzontali così da riuscire a calcolare la posizione dei piani attraverso dei calcoli tra i sensori del dispositivo uniti al tracciamento dei punti caratteristici nell'immagine.

Inoltre, questa libreria è in grado di rilevare i bordi dei piani e utilizzare tutte queste informazioni per poter posizionare degli oggetti virtuali su di essi.

Poiché ARCore basa la sua comprensione dell'ambiente su dei punti caratteristici, le superfici senza texture come un pavimento o un muro bianco potrebbero non essere rilevate facilmente.

3.3 Light estimation

Nel rendering in realtà aumentata è importante che gli oggetti virtuali siano il più possibile integrati con l'ambiente circostante. Una delle caratteristiche principali che permette all'occhio umano di percepire la posizione di un oggetto nello spazio è la luce, cioè il modo in cui esso viene illuminato e l'ombra che proietta. Proprio per questo motivo, il framework ARCore mette a disposizione il *Light estimation API*, che fornisce informazioni dettagliate riguardo l'illuminazione della scena, come spiegato nella documentazione ufficiale [7]. Tali informazioni sono necessarie per imitare i vari effetti che producono gli oggetti reali quando colpiti da una fonte di luce, che sono descritti dalla figura 3.2 nella pagina seguente:

- le ombre (*shadows*), che sono direzionali e suggeriscono dove è collocata la fonte di luce;



Fonte: <https://developers.google.com>

Figura 3.2: Esempio degli effetti prodotti dagli oggetti quando sono illuminati.

- l'ombreggiatura (*shading*), cioè l'intensità della luce che colpisce una certa faccia dell'oggetto;
- la lumeggiatura (*specular highlight*), la macchia luminosa che compare su un oggetto lucido quando viene illuminato;
- la riflessione (*reflection*), che può essere con proprietà speculari per oggetti completamente lucidi, come ad esempio uno specchio, oppure di diffusione, non dando un chiaro riflesso dell'ambiente circostante.

Le modalità per la gestione della stima della luce sono due, l'*Environmental HDR mode* e l'*Ambient intensity mode*. Durante la configurazione della sessione ARCore può essere scelta una delle due modalità, oppure disabilitare la stima della luce, come mostra il listing 3.1 nella pagina successiva tratto dalla guida ufficiale.

```

1 // Configura la sessione in modalità ENVIRONMENTAL_HDR
2 val config : Config = session.config
3 config.lightEstimationMode = LightEstimationMode.ENVIRONMENTAL_HDR
4 session.configure(config)
5
6 // Configura la sessione in modalità AMBIENT_INTENSITY
7 val config : Config = session.config
8 config.lightEstimationMode = LightEstimationMode.AMBIENT_INTENSITY
9 session.configure(config)
10
11 // Configura la sessione disabilitando la Light Estimation API
12 val config : Config = session.config
13 config.lightEstimationMode = LightEstimationMode.DISABLED
14 session.configure(config)

```

Listing 3.1: Configurazione della modalità di stima della luce.

3.3.1 Environmental HDR mode

La modalità *Environmental HDR* combina tre diverse API per replicare la luce reale, come descritti dalla figura 3.3 nella pagina seguente.

Main Directional Light Questa API calcola la direzione e l'intensità della fonte di luce principale, permettendo di posizionare correttamente l'ombra e la lumeggiatura dell'oggetto virtuale. Inoltre, questa funzionalità permette ad entrambi questi effetti ottici di venire corretti se cambia la posizione relativa dell'oggetto rispetto la fonte di luce

Ambient Spherical Harmonics Questa funzionalità permette di rappresentare la luce ambientale della scena, parametrizzando l'intensità della luce proveniente dalle varie direzioni.

HDR Cubemap Essa permette di riprodurre la riflessione di oggetti con superfici lucide. Tramite questa API viene modificata anche l'ombreggiatura e il colore dell'oggetto, che dipenderanno dalla tonalità dell'ambiente circostante.

3.3.2 Ambient intensity mode

La modalità *Ambient intensity* determina l'intensità media dei pixel e la correzione del colore di una data immagine. Dopo aver filtrato l'intensità media di un insieme di pixel e il bilanciamento del bianco per ogni frame, vengono corretti la luce e il colore dell'oggetto virtuale, affinché si integri meglio con la scena [14]. Questa modalità può essere utilizzata se la stima della luce non è critica, come per oggetti che possiedono già una propria illuminazione integrata.

Fonte:
<https://developers.googleblog.com>

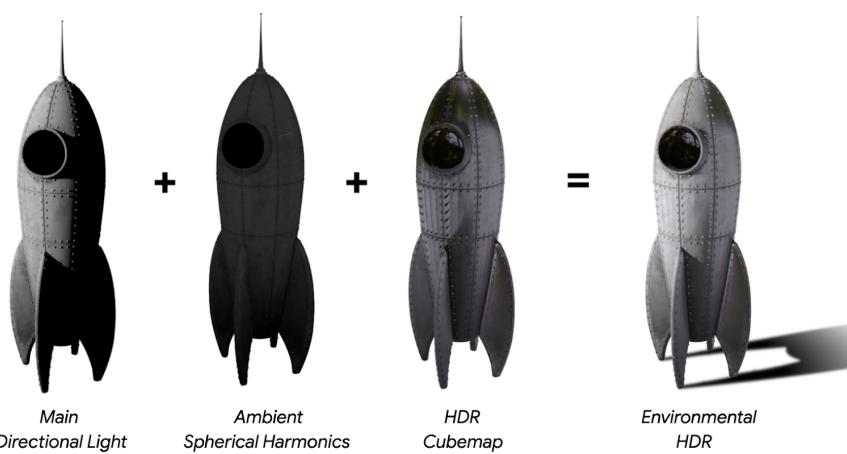


Figura 3.3: Composizione della modalità Environmental HDR.

4. Depth understanding

ARCore Depth API permette agli sviluppatori di generare mappe di profondità attraverso l'uso di algoritmi di profondità del movimento. Una mappa di profondità offre una visualizzazione in 3D del mondo reale, ogni pixel è associato alla distanza dalla scena e attraverso l'uso di colori differenti è possibile riconoscere quali aree dello spazio sono più vicine al dispositivo. Quando viene avviata una nuova sessione ARCore il display dello smartphone è nero, ma non appena si effettua un piccolo movimento la profondità viene rilevata. La stima della profondità è ricavata attraverso il movimento dello smartphone. Quando si progettano delle applicazioni che si concentrano sulla profondità bisogna considerare che la profondità viene calcolata meglio quando la scena rimane la stessa con piccoli spostamenti. Il dispositivo ha bisogno di muoversi un pò per generare la profondità. Un altro fattore da prendere in considerazione è quando l'utente compie lunghi spostamenti; in questo caso la stima della profondità arriva fino a 8 metri ma la migliore accuratezza si ha tra 0 e 5 metri.

Per ottimizzare ulteriormente le funzionalità offerte da depth API è stato effettuato un uso selettivo del machine learning Le principale funzionalità offerte da depth API sono tre:

- **Copertura dei contenuti:** permette di posizionare accuratamente dei contenuti virtuali di fronte o dietro degli oggetti reali.
- **Immersione:** permette di decorare una scena con oggetti virtuali che interagiscono tra di loro.
- **Interazione:** i contenuti virtuali sono in grado di interagire con il mondo reale attraverso cambiamenti fisici e collisioni.

4.1 Sessione ARCore con depth API

Prima di iniziare una nuova sessione ARCore è necessario controllare se il dispositivo supporta depth API. A volte questa opzione può essere disattivata oppure non supportata nonostante il dispositivo supporti ARCore. Dopo aver definito la sessione con le opportune configurazioni è possibile controllare se il dispositivo e la

fotocamera supportano una determinata modalità di profondità invocando il metodo `isDepthModeSupported(Config.DepthMode mode)` sull’istanza della sessione. Se la modalità è supportata viene configurata la sessione e sarà possibile sfruttare depth API. (Esempio ?? a pagina ??)

```

1   val config = session.config
2
3
4   // Check whether the user's device supports the Depth API.
5   val isDepthSupported = session.isDepthModeSupported(Config.DepthMode.
6     AUTOMATIC)
7   if (isDepthSupported) {
8     config.depthMode = Config.DepthMode.AUTOMATIC
9   }
10  session.configure(config)

```

Listing 4.1: Controllo supporto depth API

Per ottenere l’immagine di profondità relativa al frame corrente viene invocato il metodo `acquireDepthImage16Bits()`. (Esempio ?? a pagina ??)

```

1   val frame = arFragment.arSceneView.frame
2
3
4   // Retrieve the depth image for the current frame, if available
5   try{
6     frame.acquireDepthImage16Bits().use{ depthImage ->
7       //Use the depth image here
8     }
9   } catch(e: NotYetAvailableException){
10    // This means that depth data is not available yet.
11    // Depth data will not be available if there are no tracked
12    // feature points. This can happen when there is no motion, or when the
13    // camera loses its ability to track objects in the surrounding
14    // environment.
15  }

```

Listing 4.2: Controllo supporto depth API

4.2 Depth Hit Test

Hit Test sono stati una fondamentale interazione per le applicazioni di realtà aumentata e permettono agli utenti di posizionare oggetti 3d in una posizione precisa in una scena. Solitamente queste azioni potevano essere eseguite su superfici piane. Integrando la profondità sono stati ottenuti degli hit test più precisi grazie ai quali è possibile posizionare contenuti virtuali anche su superfici non piane in aree con bassa texture. (Esempio ?? a pagina ??)

```
1     val frame = arFragment.arSceneView.frame
2
3     val hitResultList: List<hitResult> = frame.hitTest(tap)
4
5     for(hit in hitResultList){
6         val trackable: Trackable=hit.trackable
7
8         if(trackable is Plane || trackable is Point || trackable is DepthPoint){
9             val anchor=hit.createAnchor()
10            //Use anchor here
11        }
12    }
```

Listing 4.3: Depth Hit Test

5. User Interaction

ARCore utilizza la tecnologia *ray casting* per permettere all'utente di posizionare un oggetto nella scena corrente in un punto fissato. Quando lo schermo del telefono viene toccato o viene compiuta qualche altra interazione, viene proiettato un raggio nella visuale del mondo della fotocamera che può intersecare un preciso punto o piani geometrici. ARCore permette di ricavare un elenco dei risultati delle intersezioni con la geometria della scena rilevata attraverso gli hitTest. Solitamente il primo risultato è quello più significativo perché si riferisce all'intersezione più vicina al dispositivo.

6. Oriented Points

ARCore utilizza i punti orientati quando vengono toccate superfici che non sono piane. Attorno al punto individuato dal tocco vengono esaminati dei punti caratteristici grazie ai quali è possibile stimare l'angolo dell'intersezione. Il punto orientato è costituito dal risultato del hitTest che prende considerazione questo angolo. L'invocazione del metodo `getOrientationMode()` su un oggetto Point consente di ritornare l'enumerazione `Point.OrientationMode` che restituisce la modalità di orientamento del punto.

La modalità di orientamento può essere di due tipi:

- **ESTIMATED SURFACE NORMAL** se la coordinata X è perpendicolare al raggio di proiezione e parallela alla superficie fisica centrata e attorno al hit-Test; Y giace sulla normale alla superficie stimata e Z punta verso la direzione dell'utente.
- **INITIALIZED TO IDENTITY** l'orientamento è inizializzato in base all'identità ma può variare con il tempo. Ciò che cambia dall'altra modalità è che la coordinata X punta verso la prospettiva del dispositivo dell'utente ed Y punta verso l'alto.

7. HitTest

Un hitTest è il risultato che viene restituito quando viene toccato un determinato Trackable. Ogni risultato è costituito da:

- Lunghezza in metri dall'origine del raggio che può essere ricavata dall'invocazione del metodo getDistance().
- Posa (posizione e orientamento) del punto toccato con getHitPose().
- Istanza Trackable che contiene la geometria 3d che è stata toccata con getTrackable()

Questo risultato può essere utilizzato per definire un'ancora che permette di fissare la posizione di contenuti virtuali all'interno dello spazio. L'ancora si adatta agli aggiornamenti dell'ambiente circostante e aggiorna gli oggetti legati ad essa come descritto nel capitolo (Anchor and Trackable).

Esistono quattro tipi di risultati che si possono ottenere in una sessione ARCore:

- **Profondità:** richiede l'attivazione di depth API nella sessione ARCore ed è usato per posizionare oggetti su superfici arbitrarie (non solo su piani).
- **Aereo:** permette di posizionare un oggetto su superfici piane e utilizza la loro geometria per determinare la profondità e l'orientamento del punto individuato.
- **Punto caratteristico:** permette di disporre oggetti in superfici arbitrarie basandosi su caratteristiche visive attorno al punto sul quale l'utente tocca.
- **Posizionamento istantaneo:** consente di posizionare un oggetto rapidamente in un piano utilizzando la sua geometria completa attorno al punto selezionato.

E' possibile ricevere un hitTest di tipo diverso nel seguente modo:

```

1  //Retrieve hit-test results are sorted by increasing distance from the camera or virtual ray's
2  // origin.
3  val hitResultList =
4      if (usingInstantPlacement) {
5          // When using Instant Placement, the value in APPROXIMATE_DISTANCE_METERS will determine
6          // how far away the anchor will be placed, relative to the camera's view.
7          frame.hitTestInstantPlacement(tap.x, tap.y, APPROXIMATE_DISTANCE_METERS)
8          // Hit-test results using Instant Placement will only have one result of type
9          // InstantPlacementResult.
10         } else {
11             frame.hitTest(tap)
12         }
13
14
15     // The first hit result is often the most relevant when responding to user input.
16     val firstHitResult =
17         hitResultList.firstOrNull { hit ->
18             val trackable = hit.trackable!!
19
20             if(trackable is DepthPoint){
21                 // Replace with any type of trackable type
22                 true
23             } else{
24                 false
25             }
26
27         }
28
29     if (firstHitResult != null) {
30         // Do something with this hit result. For example, create an anchor at this point of interest.
31         val anchor = firstHitResult.createAnchor()
32         //Use this anchor in your AR experience...
33     }

```

Listing 7.1: Filtraggio hitTest in base al tipo

Per definire un hitTest attraverso un raggio **arbitrario** si può usare il metodo Frame.hitTest(origin3: Array<float>, originOffset: int, direction3: Array<float>, originOffset: int) dove i quattro parametri specificano:

- origin3: array che contiene le 3 coordinate del punto di partenza del raggio.
- originOffset: offset sommato alle coordinate dell'array di partenza.
- director3: array che contiene le 3 coordinate del punto di arrivo del raggio.
- directorOffset: offset sommato alle coordinate dell'array di arrivo.

Per creare un anchor sul risultato del tocco viene usato hitResult.createAnchor() che restituirà un anchor disposto sul Trackable sottostantesu cui è venuto il tocco. Nel caso della nostra applicazione il risultato restituito da hitTest nella modalità *Plane Detection* è di tipo Aereo; il rilevamento di un piano consente di disporre un

animale in un punto preciso. Questo evento è stato gestito dal metodo *setOnTapArPlaneListener* riportato nell'esempio di codice 8.1 a pagina 18.

8. Anchor and Trackable

ARCore definisce gli Anchor per assicurare che gli oggetti virtuali rimangano nella stessa posizione e vengano tracciati nel tempo. L'ambiente circostante può cambiare, ed è necessario che la posizione di questi oggetti rimanga stabile. Gli Anchor sono disposti in insieme di punti o piani rappresentati da oggetti di tipo Trackable.

Gli oggetti Trackable rappresentano la forma geometrica sulla quale verranno definiti gli anchor. Su questi oggetti possono essere invocati 3 metodi:

- *createAnchor(*pose*: Pose)* crea un anchor in una posa che è definita nel Trackable corrente. Il tipo di oggetto Trackable definirà il modo con cui l'anchor verrà disposto e la modalità di aggiornamento della sua posa mentre il modello del mondo varia.
- *getAnchors()* restituisce tutti gli anchor presenti nel dato Trackable.
- *getTrackingState()* restituisce un oggetto TrackingState che rappresenta lo stato del Trackable. Questo stato può essere: PAUSED quando il rilevamento viene perso ma potrebbe riprendere in futuro; STOPPED quando viene fermato e non verrà più ripreso; TRACKING se viene tracciato il determinato Trackable.

La definizione di anchor e lo stato di tracciamento sono stati molto importanti per la definizione delle funzionalità principali della nostra applicazione.

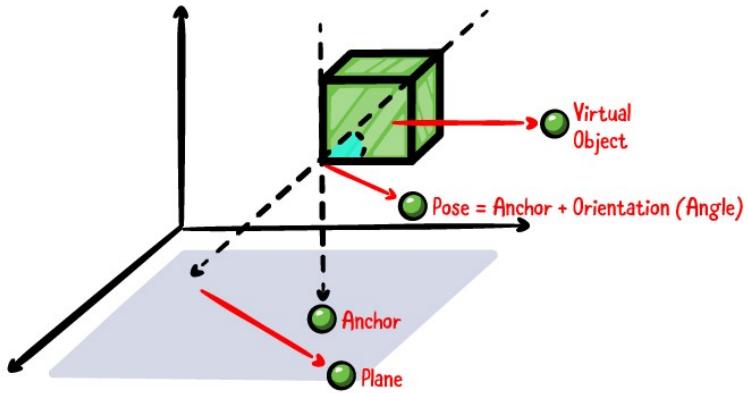
In base alla modalità l'applicazione offre funzionalità differenti:

- *Plane Detection*: ARCore rileva dei piani (Trackable) sui quali è possibile posizionare degli animali virtuali. In particolare, quando l'utente tocca un punto preciso del piano viene definito un anchor sul quale verrà renderizzato il modello 3d dell'animale. (Codice in 8.1 nella pagina successiva)
- *Augmented Images*: in ciascun frame viene controllato se lo stato di un'immagine aumentata è TRACKING; in questo caso l'immagine viene riconosciuta e viene definito un anchor nel suo centro nel quale verrà renderizzato il modello del pianeta corrispondente. (Codice in 8.2 nella pagina seguente)

Listing 8.1: Definizione Anchor in Plane Detection

```
1 //Per ogni immagine tracciata se non è presente il modello allora viene immediatamente costruito e  
2     instanziato  
3         for (augmentedImage in augmentedImages) {  
4  
5             if (augmentedImage.trackingState == TrackingState.TRACKING) {  
6  
7                 for (i in 0 until namesobj.size) {  
8  
9                     if (augmentedImage.name.contains(namesobj[i]) && !renderobj[i]) {  
10                         Toast.makeText(this, ""+namesobj[i]+" rilevato", Toast.LENGTH_SHORT)  
11                             .show()  
12  
13                         if (namesobj[i]== "systemsolar"){  
14                             renderObject(  
15                                 arFragment,  
16                                 augmentedImage.createAnchor(augmentedImage.centerPose),  
17                                 "solar_system"  
18                             )  
19                         } else {  
20                             renderObject(  
21                                 arFragment,  
22                                 augmentedImage.createAnchor(augmentedImage.centerPose),  
23                                 namesobj[i]  
24                             )  
25                         }  
26                         renderobj[i] = true  
27                     }  
28                 }  
29             }  
30         }  
31     }
```

Listing 8.2: Definizione Anchor in Augmented Images



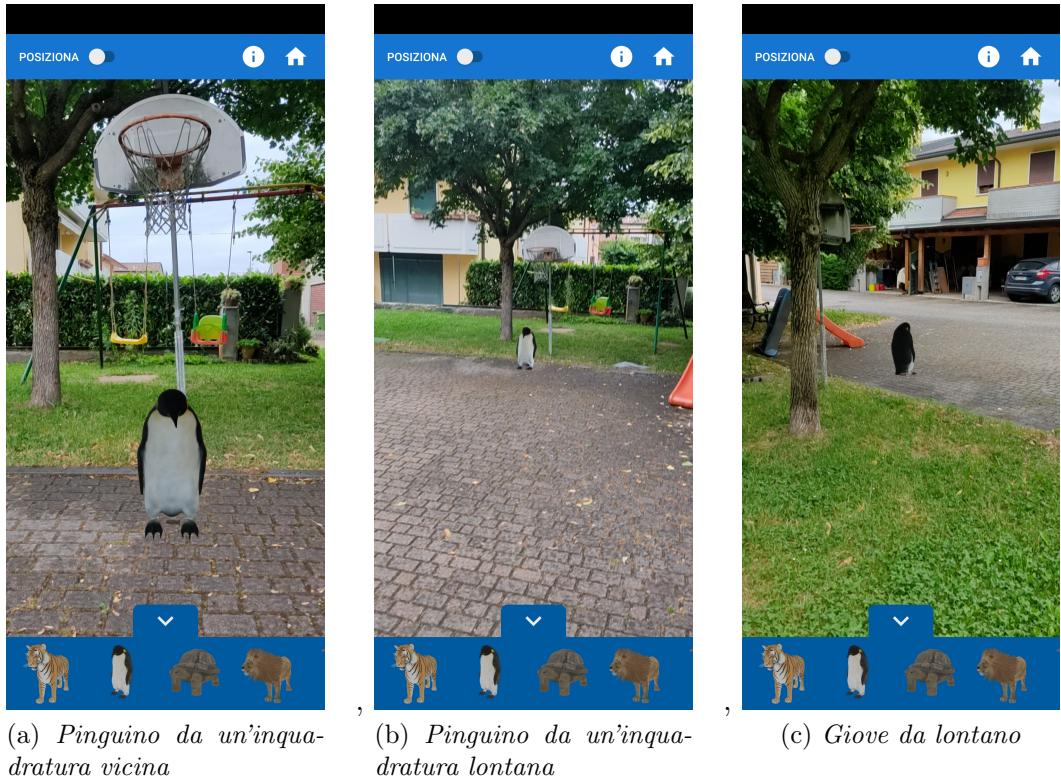
Fonte: <https://medium.com/@jaaveeth.developer/arcore-81528569eb2c>

Figura 8.1: Oggetto virtuale in un piano

Di seguito è riportata una rappresentazione di come un oggetto virtuale viene disposto in un piano.

Nella modalità *Plane Detection* la posa (posizione e orientamento) di un animale rimane invariata anche se l'ambiente circostante cambia.

Qui sotto sono riportati degli esempi in cui si può notare che il pinguino rimane nello stesso punto da qualsiasi prospettiva e distanza.



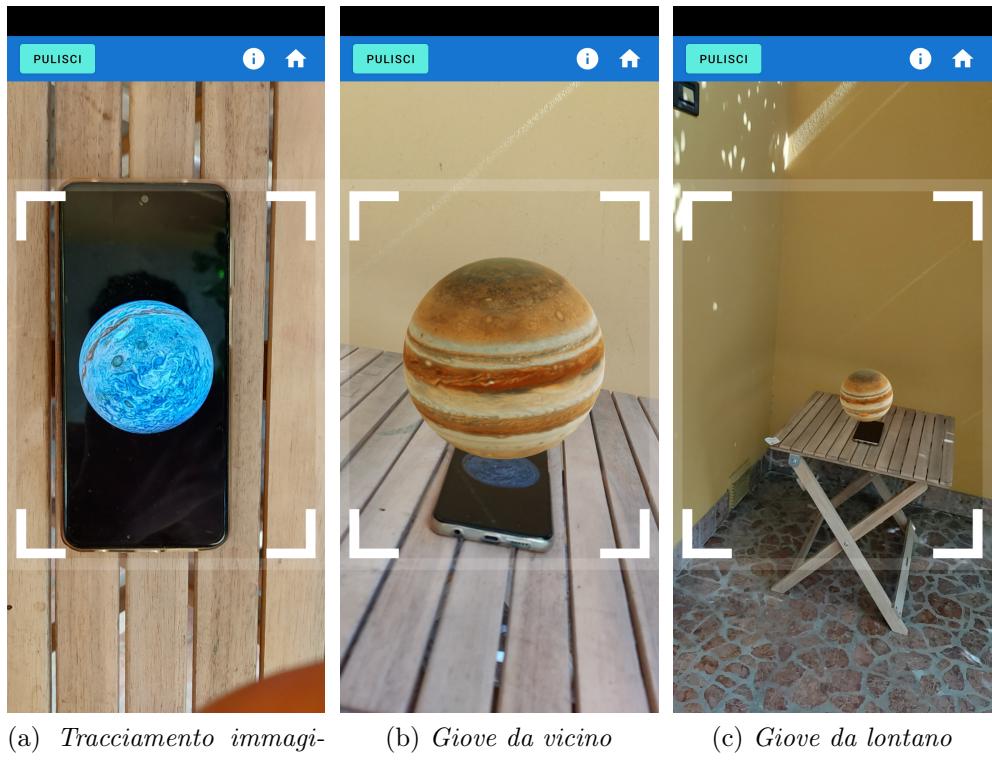
(a) Pinguino da un'inquadra-
turatura vicina

(b) Pinguino da un'inquadra-
turatura lontana

(c) Giove da lontano

Fonte: <https://developers.google.com/>

Figura 8.2: Esempio di inquadrature differenti in Plane Detection



(a) Tracciamento immagi-
ne

(b) Giove da vicino

(c) Giove da lontano

Fonte: <https://developers.google.com/>

Figura 8.3: Esempio di rilevamento in Augmented Images

9. Augmented Images

Le API Augmented Images in ARCore consentono di rilevare e poi manipolare le immagini presenti nel mondo reale e integrarle con elementi del mondo virtuale.

Dopo aver fornito delle immagini, ARCore con un algoritmo che estrae dei punti caratteristici in scala di grigi dall'immagine, ripone tutti questi dettagli in uno o più database.

Durante l'esecuzione, ARCore cerca questi punti nell'ambiente consentendo di rilevare la posizione, l'orientamento e la dimensione dell'immagine nello spazio.

9.1 Capacità

ARCore riesce a individuare fino a 20 immagini in contemporanea e non individua due istanze della stessa immagine.

Il suo database riesce a contenere fino a 1000 immagini ed inoltre, non c'è limite al numero di database che possono essere caricati. L'unico vincolo è che si può attivare ed usare un singolo database alla volta.

Quando si aggiunge un'immagine è possibile indicargli la sua grandezza nell'ambiente così da velocizzare il processo di tracciamento.

Se non viene fornita la dimensione dell'immagine, ARCore stimerà la sua grandezza e la migliorerà durante l'esecuzione. Se viene fornita la sua dimensione ARCore ignorerà la discrepanza tra la dimensione attuale e quella segnalata e scalerà tutto secondo la dimensione dell'immagine dichiarata.

ARCore è in grado di rilevare e tracciare sia immagini fisse, come ad esempio un poster nel muro, oppure immagini in movimento, come ad esempio un utente che ha l'immagine nella sua mano.

Una volta che l'immagine viene rilevata ARCore continua ad ottenere informazioni ed a rifinire la sua stima dell'immagine nell'ambiente (**FULL_TRACKING**).

Se l'immagine si muove fuori dall'inquadratura ARCore continua a stimare la sua posizione assumendo che l'immagine sia statica e non si muova nell'ambiente (**LAST_KNOWN_POSE**).

9.2 Requisiti

L'immagine deve:

- Essere visibile almeno al 25% nell'inquadratura
- Essere piatta e non accartocciata
- Essere libera da ogni oggetto tra la fotocamera e l'immagine. Non deve essere oscurata, vista troppo angolata oppure vista quando la fotocamera si muove troppo velocemente

9.3 Utilizzo della CPU e considerazioni sulle performance

In base alle funzionalità di ARCore abilitate, la batteria potrebbe scaricarsi più velocemente e può aumentare l'utilizzo della CPU. E' consigliato disabilitare le funzionalità non utilizzate per preservare la batteria e rendere più leggero il carico della CPU.

9.4 Creazione del database

Esistono due metodi per creare un database di immagini che verranno successivamente usate da ARCore.

Il primo metodo è importare un database creato in precedenza con il tool *arcoreimg*. Per importarlo si procede come descritto dal listing 9.1, tratto dalla guida ufficiale.

```

1  val imageDatabase = this.assets.open("example.imgdb").use {
2
3      AugmentedImageDatabase.deserialize(session, it)
4 }
```

Listing 9.1: Descrizione del listing.

Il secondo metodo consiste nel crearlo a runtime partendo con delle immagini contenuti nella cartella assets dell'applicazione, come descritto dal listing 9.2 nella pagina seguente.

```
1 val imageDatabase = AugmentedImageDatabase(session)
2
3 val bitmap = assets.open("dog.jpg").use { BitmapFactory.decodeStream(it) }
4
5 // If the physical size of the image is not known, use addImage(String, Bitmap) instead, at
6 // the expense of an increased image detection time.
7
8 val imageWidthInMeters = 0.10f // 10 cm
9
10 val dogIndex = imageDatabase.addImage("dog", bitmap, imageWidthInMeters)
```

Listing 9.2: Descrizione del listing.

9.5 Come abilitare il tracciamento delle immagini

Per abilitare il tracciamento delle immagini bisogna modificare la configurazione della sessione per impostare il database da utilizzare, come descritto dal listing 9.3

```
1 val config = Config(session)
2
3 config.augmentedImageDatabase = imageDatabase
4
5 session.configure(config)
```

Listing 9.3: Descrizione del listing.

10. Instant Placement

L'API di posizionamento istantaneo consente di posizionare gli oggetti nell'ambiente prima ancora che venga spostato il dispositivo e che vengano individuati i piani.

Dopo che l'utente ha posizionato l'oggetto, la sua posizione viene perfezionata in tempo reale mentre l'utente si muove nell'ambiente.

Successivamente, quando ARCore è in grado di identificare con precisione l'ubicazione dell'oggetto nello spazio, viene cambiata la sua posizione e dimensione per rispettare la scala dell'ambiente.

10.1 Abilitare Instant Placement

Per abilitare l'instant placement bisogna creare una sessione configurata per supportare l'instant placement, come descritto dal listing 10.1

```
1 fun createSession() {  
2     val session = Session(applicationContext);  
3     val config = Config(session)  
4     // Set the Instant Placement mode.  
5     config.instantPlacementMode = Config.InstantPlacementMode.LOCAL_Y_UP  
6     session.configure(config)  
7 }
```

Listing 10.1: Descrizione del listing.

11. Recording Playback

Solitamente quando si utilizza un'applicazione AR ci si trova ad essere in prima persona nell'ambiente il cui deve essere utilizzata.

L'API Recording and Playback consente di utilizzare i servizi di ARCore anche su un feed video non in tempo reale. Più precisamente si può fornire un video e ARCore lo tratterà come se fosse un video registrato in tempo reale.

L'API Recording archivia il video stream, i dati IMU o qualsiasi altri metadati personalizzati che scegli di salvare in un file MP4. Successivamente l'utente potrà scegliere se usare un video dal vivo oppure un video preregistrato.

11.1 Compatibilità

ARCore è necessario per utilizzare API Recording and Playback perché serve per registrare i dati nel file MP4. I file MP4 registrati, utilizzando questa funzionalità, sono essenzialmente file video con dati aggiuntivi e possono essere visualizzati utilizzando qualsiasi video player. Per ispezionare i seguenti file è possibile usare, per esempio, il player Exoplayer di Android.

11.2 Come vengono registrati i dati video e AR per la riproduzione

ARCore salva le registrazioni in formato MP4. All'interno di questo file sono contenuti più tracciati video registrati con la codifica H.264 e dati vari.

La prima traccia video viene solitamente registrata ad una risoluzione di 640x480 (VGA) e questa traccia verrà usata per il motion tracking come fonte di video primaria.

Nel caso si voglia una risoluzione migliore, bisogna configurare una nuova fotocamera che abbia la risoluzione desiderata.

In questo caso ARCore richiederà un feed in qualità 640x480 e un feed in alta risoluzione. Questo potrebbe rallentare la propria applicazione a causa di un maggiore utilizzo della CPU. Verrà inoltre selezionata la risoluzione personalizzata come fonte primaria del video che verrà salvato nel file MP4.

La seconda traccia del file MP4 è una visualizzazione della mappa della profondità della fotocamera. Si tratta di un video ricavato dal sensore di profondità del proprio dispositivo e successivamente convertito in valori dei canali RGB.

11.3 Ulteriori dati salvati

ARCore registra anche le misurazioni del giroscopio e dell'accelerometro del dispositivo. Inoltre, vengono salvati anche altri dati, tra cui alcuni considerati sensibili. Questi dati sono la versione SDK di Android, il fingerprint del dispositivo, informazioni addizionali sui sensori usati e, se il ARCore Geospatial API è attivo , la posizione stimata, i dati del magnetometro e della bussola.

11.4 Come registrare

Per iniziare una sessione di registrazione inizialmente creo una sessione di ARCore, imposto il file di output e alcune configurazioni sulla registrazione. Infine inizio a registrare. Per concludere la registrazione è necessario invocare il metodo `stopRecording()` sull'istanza `session`. Si veda il listing 11.1 tratto dalla documentazione ufficiale Google.

```

1 // Configure the ARCore session.
2 val session = Session(context)
3
4 val destination = Uri.fromFile(File(context.getFilesDir(), "recording.mp4"))
5
6 val recordingConfig = RecordingConfig(session)
7     .setMp4DatasetUri(destination)
8     .setAutoStopOnPause(true)
9
10 session.startRecording(recordingConfig)
11
12 // Resume the ARCore session to start recording.
13 session.resume()
14
15 // Stop recording
16 session.stopRecording()
```

Listing 11.1: Descrizione del listing.

12. Augmented faces

L'API *Augmented Faces* permette di identificare i volti umani e le varie parti che lo compongono tramite Intelligenza Artificiale, per sovrapporre ad essi modelli 3D come maschere, occhiali, cappelli utilizzando solo la fotocamera frontale [5]. Questa libreria permette ottenere un *face mesh*, una rappresentazione virtuale composta da una maglia di punti che riproduce il profilo del volto [12]. Oltre ad essa, l'API fornisce un *center pose* e tre *region pose*, come descritti dalla figura 12.1 nella pagina seguente tratte dalla documentazione ufficiale.

Face mesh Consiste in una rete di 468 punti, che permette di posizionare una texture sul volto. Essa viene tracciata come un piano, per permettere all'immagine virtuale di seguire il volto anche se in movimento, come spiegato in [8].

Center pose Rappresenta il centro del volto, posizionato dietro il naso. Utile per il rendering di oggetti virtuali da posizionare sopra la testa.

Region pose Identifica una regione rilevante del volto, come i lati destro o sinistro della fronte, oppure il naso. Sono utili per il rendering di oggetti virtuali da posizionare sul naso o attorno agli orecchi.

12.1 Configurazione e utilizzo

La configurazione della sessione ARCore deve essere effettuata selezionando la fotocamera frontale ed abilitando la modalità Augmented Face, come mostrato nel listing 12.1 nella pagina successiva tratto dalla guida ufficiale Google.

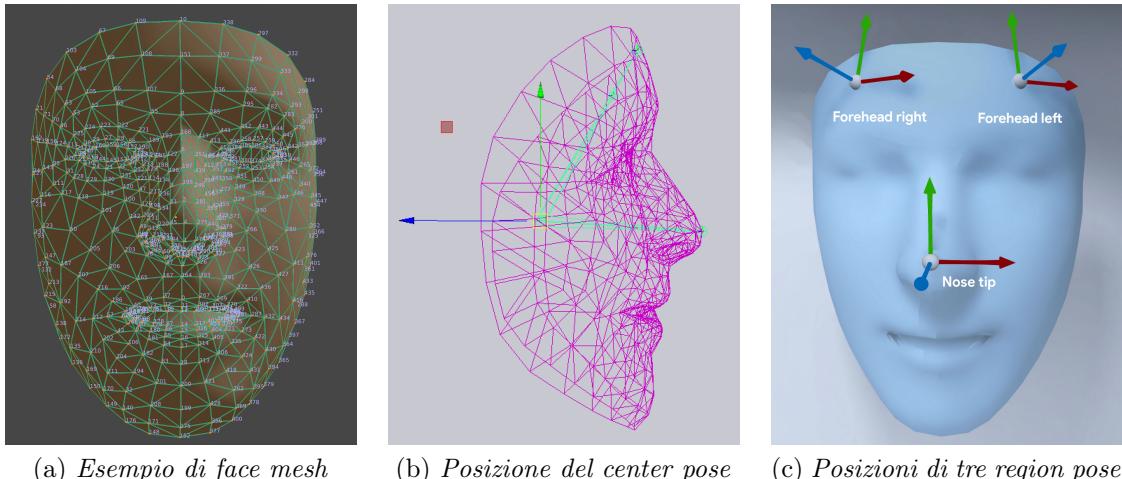


Figura 12.1: Elementi ottenuti tramite l'API Augmented Faces

```

1 // Configura la sessione utilizzando la camera frontale.
2 val filter = CameraConfigFilter(session).setFacingDirection(CameraConfig.
3   FacingDirection.FRONT)
4 val cameraConfig = session.getSupportedCameraConfigs(filter)[0]
5 session.cameraConfig = cameraConfig
6
7 // Abilita la modalità Augmented Face.
8 val config = Config(session)
9 config.augmentedFaceMode = Config.AugmentedFaceMode.MESH3D
10 session.configure(config)

```

Listing 12.1: Configurazione della modalità Augmented Face.

Da ogni frame è possibile ricavare un oggetto **Trackable**, che può essere tracciato e a cui possono essere collegate degli **Anchor**. Verificando lo stato di ogni oggetto **Trackable** restituito, è possibile ricavare i region pose, il center pose e i vertici del face mesh, per poi procedere con il rendering degli oggetti virtuali. Si veda il listing 12.2 nella pagina seguente tratto dalla documentazione ufficiale per un possibile utilizzo.

```
1 // Ricava gli oggetti trackable dalla sessione ARCore
2 val faces = session.getAllTrackables(AugmentedFace::class.java)
3
4 // Verifica lo stato di ogni oggetto contenuto nella lista di Trackable
5 faces.forEach { face ->
6     if (face.trackingState == TrackingState.TRACKING) {
7         // Ricava il center pose
8         val facePose = face.centerPose
9
10        // Ricava i region pose
11        val foreheadLeft = face.regionPose(AugmentedFace.RegionType.FOREHEAD_LEFT)
12        val foreheadRight=face.regionPose(AugmentedFace.RegionType.FOREHEAD_RIGHT)
13        val noseTip = face.regionPose(AugmentedFace.RegionType.NOSE_TIP)
14
15        // Ricava i vertici del face mesh
16        val faceVertices = face.meshVertices
17
18        // Rendering dell'oggetto virtuale
19    }
20}
```

Listing 12.2: Utilizzo della modalità Augmented Faces.

13. Cloud anchors

L'API ARCore *Cloud Anchor* introduce i cloud anchor, un tipo speciale di anchor che permettono di condividere con altri utenti l'esperienza AR. In particolare, un dispositivo può posizionare oggetti virtuali nello spazio, e altri utenti possono vedere l'oggetto virtuale ed interagire con esso trovandosi nella stessa posizione, come spiegato da [11].

Questo tipo di anchor, come spiegato dalla documentazione ufficiale [6], trova applicazione ad esempio per creare oggetti virtuali che persistano nel mondo reale, cioè che mantengano nel tempo la posizione in cui sono stati creati, oppure per creare giochi virtuali multigiocatore in cui è importante la collaborazione tra utenti in tempo reale.

13.1 Funzionamento

La creazione e la diffusione dell'anchor avviene tramite connessione internet in quattro passaggi, descritti ad alto livello come segue. La figura 13.1 nella pagina successiva tratta dalla guida ufficiale Google descrive visivamente le quattro fasi del funzionamento dei cloud anchor.

1. **Creation:** l'utente crea un anchor localmente;
2. **Hosting:** ARCore carica i dati della mappa 3D dello spazio circostante all'anchor locale nell'ARCore Cloud Anchor, che a sua volta restituisce al dispositivo un Cloud Anchor ID univoco;
3. **Distribution:** l'app distribuisce l'ID univoco agli altri utenti;
4. **Resolving:** gli utenti possono utilizzare l'ID ricevuto per ricreare l'anchor quando si trovano nello stesso ambiente. L'API compara la scena del dispositivo con la mappa 3D caricata precedentemente per determinare la posizione dell'utente e visualizzare correttamente l'oggetto virtuale.

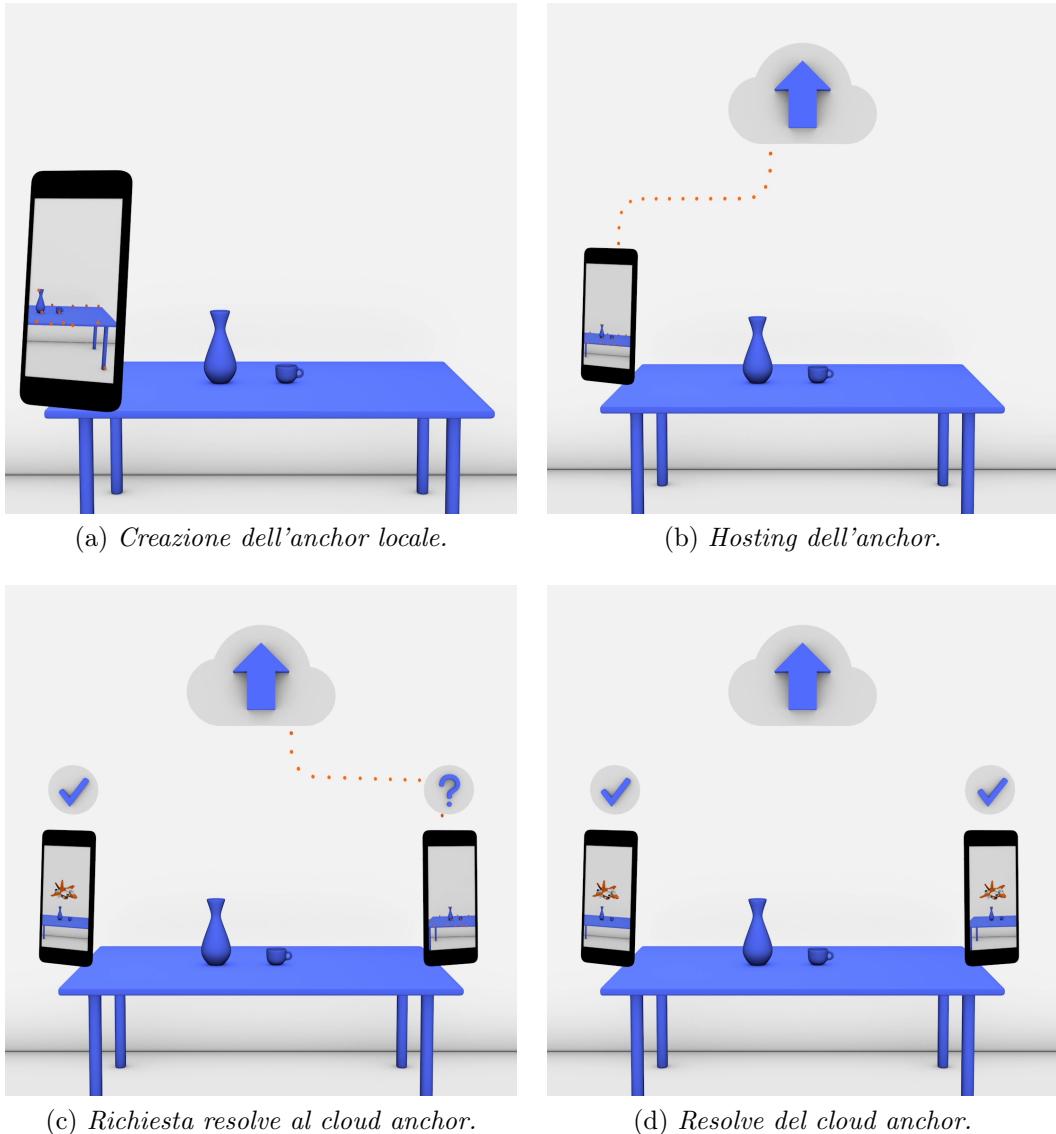


Figura 13.1: Funzionamento ad alto livello di Cloud Anchor API.

13.2 Implementazione e utilizzo

Per implementare un'applicazione che utilizzi i Cloud Anchor è prima necessario creare un progetto in *Google Cloud Platform* e abilitare il ARCore Cloud Anchor API per l'hosting, il salvataggio e il resolving degli anchor.

La sessione ARCore dell'app deve poter comunicare con il progetto cloud creato, come descritto dal listing 13.1 tratto dalla documentazione ufficiale.

```
1 val config = Config(session)
2 config.cloudAnchorMode = Config.CloudAnchorMode.ENABLED
3 session.configure(config)
```

Listing 13.1: Configurazione della modalità Cloud Anchor.

Autenticazione...

14. Geospatial API

Bibliografia

- [1] Ronald T. Azuma. «A Survey of Augmented Reality». In: *Presence: Teleoperators and Virtual Environments* 6.4 (ago. 1997), pp. 355–385. DOI: [10.1162/pres.1997.6.4.355](https://doi.org/10.1162/pres.1997.6.4.355).
- [2] Bimber et al. *Spatial Augmented Reality Merging Real and Virtual Worlds*. Ago. 2005. ISBN: 9780429108501. DOI: [10.1201/b10624](https://doi.org/10.1201/b10624).
- [3] Julie Carmigniani et al. «Augmented Reality Technologies, Systems and Applications». In: *Multimedia Tools Appl.* 51.1 (gen. 2011), pp. 341–377. ISSN: 1380-7501. DOI: [10.1007/s11042-010-0660-6](https://doi.org/10.1007/s11042-010-0660-6).
- [4] Adrian David Cheok et al. «Human Pacman: A Sensing-Based Mobile Entertainment System with Ubiquitous Computing and Tangible Interaction». In: *Proceedings of the 2nd Workshop on Network and System Support for Games*. NetGames '03. Redwood City, California: Association for Computing Machinery, 2003, pp. 106–117. ISBN: 1581137346. DOI: [10.1145/963900.963911](https://doi.org/10.1145/963900.963911).
- [5] Google developers. *Augmented Faces introduction*. Feb. 2022. URL: <https://developers.google.com/ar/develop/augmented-faces>.
- [6] Google developers. *Cloud Anchors allow different users to share AR experiences*. Apr. 2022. URL: <https://developers.google.com/ar/develop/cloud-anchors>.
- [7] Google developers. *Get the lighting right*. Mag. 2022. URL: <https://developers.google.com/ar/develop/lighting-estimation>.
- [8] Google developers. *New UI tools and a richer creative canvas come to ARCore*. Feb. 2019. URL: <https://developers.googleblog.com/2019/02/new-ui-tools-and-richer-creative-canvas.html>.
- [9] George W. Fitzmaurice. «Situated Information Spaces and Spatially Aware Palmtop Computers». In: *Commun. ACM* 36.7 (1993), pp. 39–49. ISSN: 0001-0782. DOI: [10.1145/159544.159566](https://doi.org/10.1145/159544.159566).
- [10] Anders Henrysson, Mark Billinghurst e Mark Ollila. «AR Tennis». In: SIGGRAPH '06 (2006), 1–es. DOI: [10.1145/1179133.1179135](https://doi.org/10.1145/1179133.1179135).

-
- [11] Ida Bagus Kerthyayana Manuaba. «Mobile based Augmented Reality Application Prototype for Remote Collaboration Scenario Using ARCore Cloud Anchor». In: *Procedia Computer Science* 179 (2021). 5th International Conference on Computer Science and Computational Intelligence 2020, pp. 289–296. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2021.01.008>.
 - [12] Zainab Oufqir, Abdellatif El Abderrahmani e Khalid Satori. «ARKit and ARCore in serve to augmented reality». In: *2020 International Conference on Intelligent Systems and Computer Vision (ISCV)*. 2020, pp. 1–7. DOI: 10.1109/ISCV49265.2020.9204243.
 - [13] Jun Rekimoto e Katashi Nagao. «The World through the Computer: Computer Augmented Interaction with Real World Environments». In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. UIST '95. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1995, pp. 29–36. ISBN: 089791709X. DOI: 10.1145/215585.215639.
 - [14] Aleksi Suonsivu. «RGBD SLAM Based 3D Object Reconstruction and Tracking: Using Google ARCore». B.S. thesis. 2020.
 - [15] Ivan E. Sutherland. «A Head-Mounted Three Dimensional Display». In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. AFIPS '68 (Fall, part I). San Francisco, California: Association for Computing Machinery, 1968, pp. 757–764. ISBN: 9781450378994. DOI: 10.1145/1476589.1476686.
 - [16] B. Thomas et al. «ARQuake: an outdoor/indoor augmented reality first person application». In: *Digest of Papers. Fourth International Symposium on Wearable Computers*. 2000, pp. 139–146. DOI: 10.1109/ISWC.2000.888480.