

Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Google ARCore

Marco Vettore
Matricola 1216433

Alberto Varini
Matricola 1227153

Mattia Tamiazzo
Matricola 1219603

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	Light estimation	7
4	Depth understanding	10
4.1	Depth API	10
4.2	Raw Depth API	13
4.3	Depth Hit Test	15
4.4	Depth Lab	16
5	User Interaction	19
6	Oriented Points	20
7	HitTest	21
7.1	Definizione e gestione di un HitTest	22
8	Anchor and Trackable	24
9	Augmented faces	28
9.1	Configurazione e utilizzo	29
10	Cloud anchors	30
10.1	Funzionamento	30
10.2	Configurazione e utilizzo	31
11	Geospatial API	33
11.1	Configurazione e utilizzo	33

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 [27] 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* [16], 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* [24], 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* [28] 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* [18], 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 di un 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* 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 (dati non visuali) come la velocità, accelerazione e posizione. La rilevazione di punti caratteristici è l'individuazione di immagini con caratteristiche differenti (dati visuali) 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 [13]. 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.

Nella figura 3.1 nella pagina seguente si può notare come la posizione dell'utente è tracciata in relazione ai punti caratteristici identificati nel divano reale.

Nella figura 3.2 a pagina 6 sono rappresentati i moduli principali dello SLAM che si differenziano in quattro categorie [19]:

- *Sensori*: nel caso degli smartphone fotocamera, sensore IMU (accelerometro,



Fonte: <https://developers.google.com/ar/develop/fundamentals>

Figura 3.1: Motion Tracking.

giroscopio). Potrebbero essere inclusi altri sensori per migliorare la precisione come GPS, sensori di luce, profondità.

- *Front-end*: riceve i dati (visuali e non visuali) che permettono di ricavare una stima della posa. Dai dati *visuali* identifica i punti caratteristici dai quali vengono estratti i *descrittori di features*; mentre da quelli *non visuali* ricava una stima precisa della posa correlando le caratteristiche spaziali di un frame con quelle osservate in sequenze di frame. I descrittori di features descrivono orientazione, scala, gravità, direzione e altri aspetti. Questo modulo include il riconoscimento dei luoghi già visitati se le stesse features vengono associate molte volte ai punti caratteristici.
- *Back-end*: elabora l'output del front-end dal quale crea una rappresentazione 3d dell'ambiente sulla base di una pluralità memorizzata di mappe, descrittori di features e dalla stima della posa del dispositivo. Successivamente passa questa ricostruzione geometrica alla SLAM estimate.
- *SLAM estimate*: calcola una posa localizzata cercando di ridurre al minimo le discrepanze tra i descrittori di features estratti e memorizzati nel tempo.

Nell'esempio (a) in figura 3.4 nella pagina seguente si può vedere come il margine di errore si accumula nel tempo portando ad una rappresentazione inconsistente della mappa dell'ambiente. In (b) si può notare un miglioramento della mappa allineando le varie scansioni in base ai vincoli imposti dalle pose relative.

Per eseguire dei test sulle **performance** del Motion Tracking si devono tenere in considerazione alcuni aspetti:

- *Angolazione*: l'abilità di rilevare punti caratteristici o superfici può dipendere dall'angolazione del dispositivo. Con alcuni angoli si potrebbero ottenere risultati migliori.

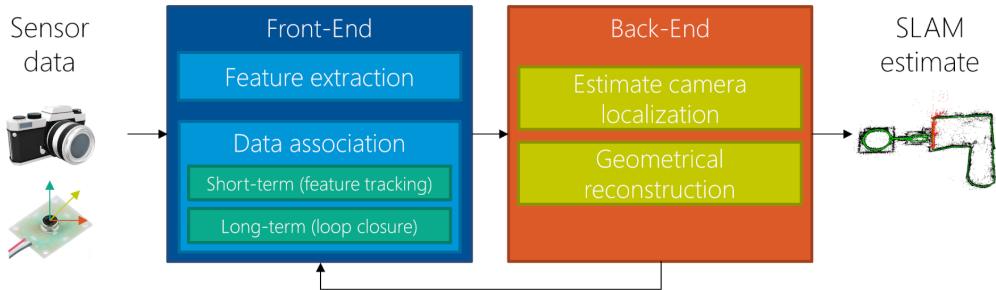


Diagram based on: Cadena, Cesar, et al. "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age." *IEEE Transactions on Robotics* 32.6 (2016): 1309–1332.
 SLAM estimate: R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," in *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163, Oct. 2015.
 3D Model of motion sensor by gerduleb: <https://www.remix3d.com/details/G0095VNP5RSM>
 3D Model of camera by Microsoft: <https://www.remix3d.com/details/G0095XQ93TH9>

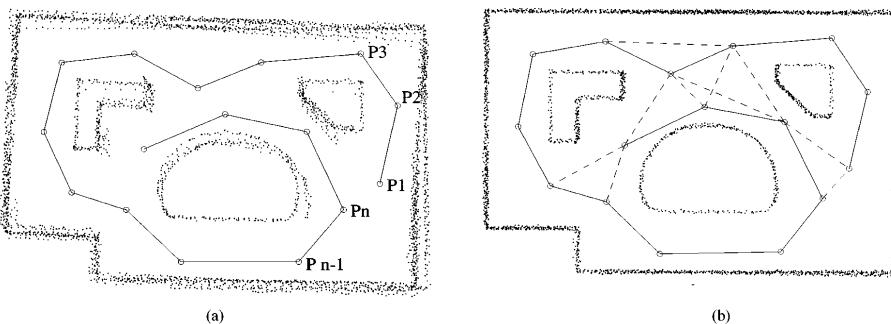
Fonte: <https://www.andreasjakl.com/basics-of-ar-slam-simultaneous-localization-and-mapping/>

Figura 3.2: Struttura dei moduli SLAM



Fonte: <https://www.youtube.com/watch?v=UkIcupOTJrY&t=170s>

Figura 3.3: Punti caratteristici.



Fonte: <https://www.andreasjakl.com/basics-of-ar-slam-simultaneous-localization-and-mapping/>

Figura 3.4: Correzione degli errori da una mappa.

- *Movimento*: questo test potrebbe essere iniziato con un movimento lento e con un aumento progressivo. A seguito di un movimento più rapido il dispositivo dovrebbe essere in grado di comprendere l'ambiente circostante e tracciare eventuali contenuti virtuali. Altri test possono essere effettuati con movimenti rapidi improvvisi e la copertura della fotocamera.

3.2 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 [8]. 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 3.5 nella pagina successiva:

- le **ombre** (*shadows*), che sono direzionali e suggeriscono dove è collocata la fonte di luce;
- 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 seguente tratto dalla guida ufficiale.



Figura 3.5: Esempio degli effetti prodotti dagli oggetti quando sono illuminati.

```

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.

3.2.1 Environmental HDR mode

La modalità *Environmental HDR* combina tre diverse API per replicare la luce reale, come descritti dalla figura 3.6 nella pagina successiva.

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

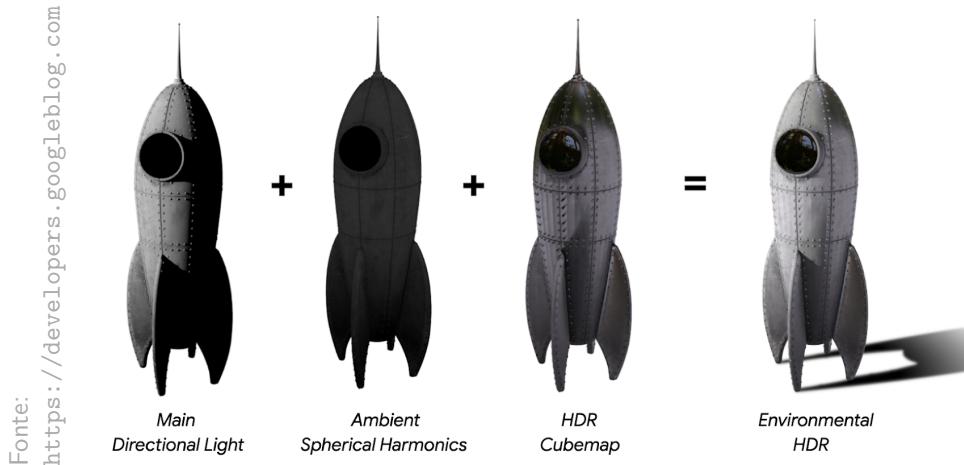


Figura 3.6: Composizione della modalità Environmental HDR.

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.

3.2.2 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 [26]. Questa modalità può essere utilizzata se la stima della luce non è critica, come per oggetti che possiedono già una propria illuminazione integrata.

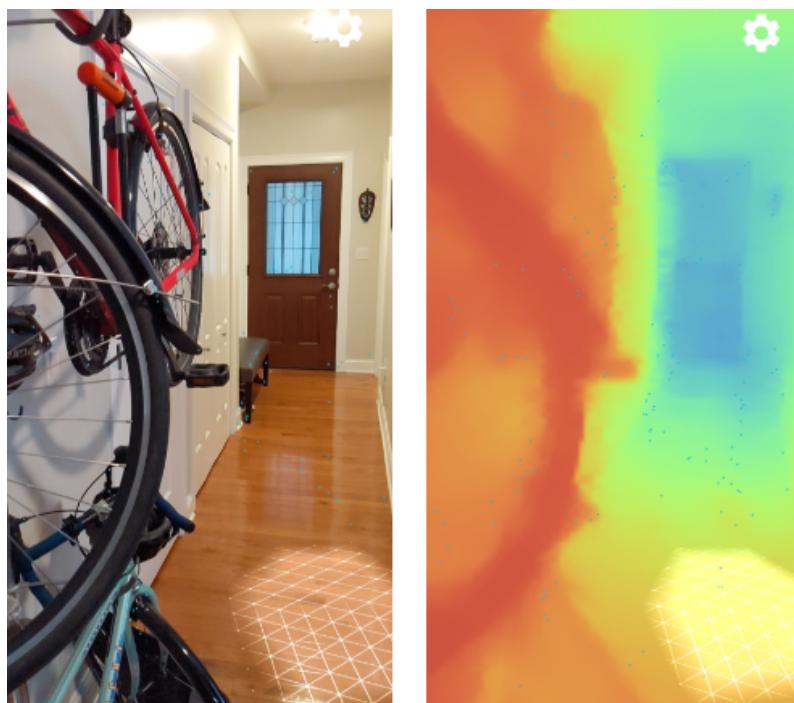
4. Depth understanding

4.1 Depth API

ARCore Depth API aggiunge un livello di realismo attraverso l'uso di algoritmi che generano immagini o mappe di profondità. Questi algoritmi sono in grado di ottenere stime di profondità fino a 65 metri. Un'immagine di profondità (figura 4.1 nella pagina seguente) offre una visualizzazione 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. La profondità viene acquisita con piccoli spostamenti dai quali si ottengono misure più efficienti (fino a 5 metri). Per ciascun frame può essere recuperata l'immagine di profondità corrispondente dopo che l'utente abbia iniziato a muoversi con il dispositivo. Depth API richiede una grande potenza di elaborazione ed è supportata solo da alcuni dispositivi; è necessario controllare se il dispositivo è compatibile e nel caso lo fosse attivare la profondità manualmente nella configurazione della sessione ARCore.

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.



Fonte: <https://developers.google.com/ar/develop/depth>

Figura 4.1: Esempio di mappa di profondità.

4.1.1 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 [10]. Si veda il listing 4.1 per un esempio di configurazione.

```

1 val config = session.config
2
3 // Verifica se il dispositivo supporta Depth API.
4 val isDepthSupported = session.isDepthModeSupported(Config.DepthMode.AUTOMATIC)
5 if (isDepthSupported) {
6     config.depthMode = Config.DepthMode.AUTOMATIC
7 }
8 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()`, come mostrato dal listing 4.2.

```

1 val frame = arFragment.arSceneView.frame
2
3 // Estrai il depth image per il frame corrente, se disponibile.
4 try{
5     frame.acquireDepthImage16Bits().use{ depthImage ->
6         // Uso del depth image
7     }
8 } catch(e: NotYetAvailableException){
9     // I dati sulla profondità non sono disponibili.
10    // I dati sulla profondità non sono disponibili se non ci sono feature point
11    // tracciati. Questo può succedere se non c'è movimento o quando la fotocamera
12    // non è più in grado di tracciare gli oggetti dell'ambiente circostante.
13 }

```

Listing 4.2: Estrazione di un'immagine profonda.

```

1 // Restituisce la profondità in millimetri di depthImage alle coordinate ([x], [y]).  

2 fun getMillimetersDepth(depthImage:Image, x:Int, y:Int): Int {  

3  

4     // Il depth image ha un singolo piano, che salva la profondità per ogni pixel  

5     // in una variabile unsigned integer a 16 bit.  

6     val plane = depthImage.planes[0]  

7  

8     // Distanza in byte tra campioni di pixel adiacenti.  

9     val pixelStride = x * plane.pixelStride  

10  

11    // Row stride in byte per il piano.  

12    val rowStride = y * plane.rowStride  

13  

14    val byteIndex = pixelStride + rowStride  

15  

16    // Recupera l'ordine in byte del buffer.  

17    val buffer = plane.buffer.order(ByteOrder.nativeOrder())  

18  

19    // Leggi due byte all'indice dato, componendoli in un valore piccolo  

20    // in base al corrente ordine in byte.  

21    val depthSample = buffer.getShort(byteIndex)  

22  

23    return depthSample.toInt()  

24 }

```

Listing 4.3: Estrazione di informazioni da un’immagine profonda.

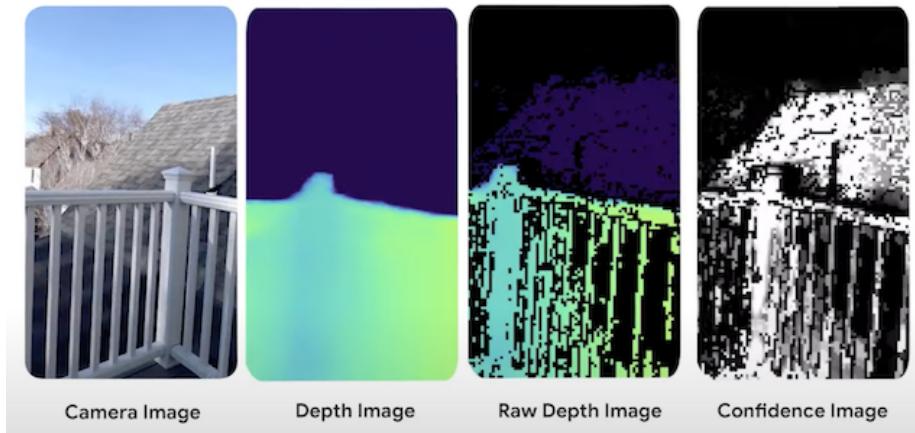
4.2 Raw Depth API

Le *ARCore Raw Depth API* forniscono informazioni più precise sulla profondità di alcuni pixel di un’immagine e permettono di rappresentare la geometria della scena generando delle **immagini di profondità grezze**. Ad esse vengono associate delle **immagini di affidabilità** che stabiliscono il grado di confidenza di ciascun pixel (associato ad un valore di profondità) [22]. In particolare, ogni pixel è un numero intero senza segno a 8 bit che indica la stima della confidenza con valori compresi tra 0 (più bassa) e 255 (più alta) inclusi. I pixel senza una stima della profondità valida hanno un valore di confidenza pari a 0 e un valore di profondità nullo.

L’uso di Raw Depth API è molto utile nei casi in cui c’è il bisogno di avere una maggiore precisione, ad esempio nella **misurazione (PHORIA ARConnect App)**, **ricostruzione 3d (3d Live Scanner App)**, **rilevamento delle forme (Jam3)**, mostrati nella figura 4.2 nella pagina successiva.

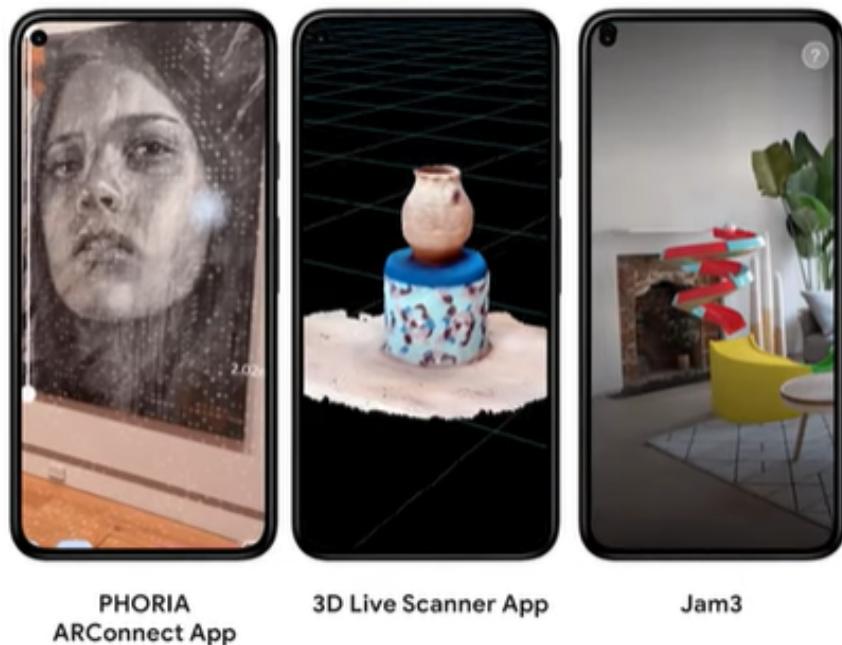
4.2.1 Sessione ARCore con Raw depth API

Inizialmente si deve effettuare il controllo sulla compatibilità del dispositivo come riportato nel codice 4.1 nella pagina precedente. Per acquisire un’immagine di confidenza viene invocato il metodo `acquireRawDepthConfidenceImage()`, dalla quale si può ricavare l’accuratezza di ogni pixel di profondità non elaborato [11]. Si veda il listing 4.4 a pagina 15.



Fonte: <https://stackoverflow.com/questions/59088045/arcore-raw-depth-data-from-rear-android-depth-camera>

Figura 4.2: Differenze tra immagini in API Raw Depth.



Fonte: <https://www.youtube.com/watch?v=13WugTM0dSs>

Figura 4.3: Applicazioni Raw Depth API

```

1  try {
2      // All'inizio, recupera un raw depth image.
3      // Un Raw Depth image è un uint16, con GPU aspect ratio e orientamento nativo.
4      frame.acquireRawDepthImage16Bits().use { rawDepth ->
5
6          // Il Confidence image è un uint8, in corrispondenza alla dimensione di depth image.
7          frame.acquireRawDepthConfidenceImage().use { rawDepthConfidence ->
8
9              // Compara i timestamp per determinare se la distanza è basata su nuovi
10             // depth data, o se è una proiezione basata sul movimento del dispositivo.
11             val thisFrameHasNewDepthData = frame.timestamp == rawDepth.timestamp
12
13             if (thisFrameHasNewDepthData) {
14                 val depthData = rawDepth.planes[0].buffer
15                 val confidenceData = rawDepthConfidence.planes[0].buffer
16                 val width = rawDepth.width
17                 val height = rawDepth.height
18                 someReconstructionPipeline.integrateNewImage(
19                     depthData,
20                     confidenceData,
21                     width = width,
22                     height = height
23                 )
24             }
25         }
26     }
27 } catch (e: NotYetAvailableException) {
28     // Depth image non è ancora disponibile.
29 }
```

Listing 4.4: Estrazione di un'immagine di confidenza.

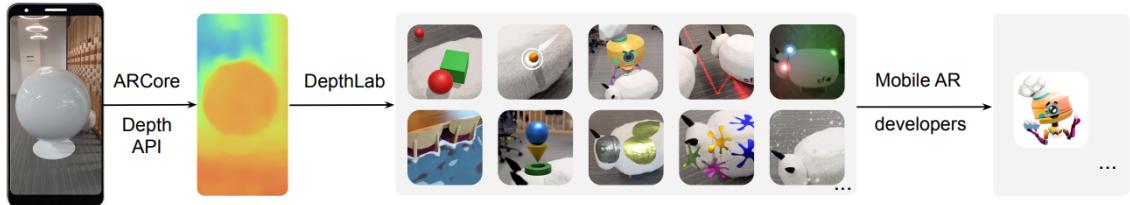
4.3 Depth Hit Test

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 [10]. Si veda il listing 4.5.

```

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         // Uso di anchor...
11     }
12 }
```

Listing 4.5: Depth Hit Test



Fonte: https://augmentedperception.github.io/depthlab/assets/Du_DepthLab-Real-Time3DIteractionWithDepthMapsForMobileAugmentedReality_UIST2020.pdf

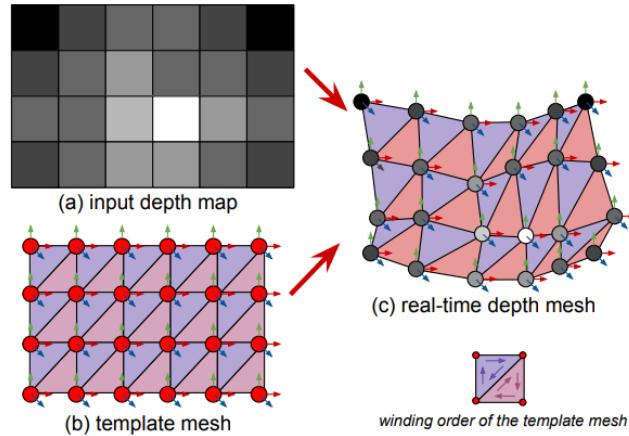
Figura 4.4: Panoramica ad alto livello di Depth Lab

4.4 Depth Lab

Depth Lab è una libreria software che incapsula un insieme ristretto di depth API grazie alle quali è possibile recuperare risorse utili per **rilevamento** della geometria e **rendering** nell’interazione AR. Si veda la figura 4.4.

Depth Lab è costituito da quattro componenti [14]:

- **Tracking and Input:** vengono utilizzate immagini di profondità (ottenute da Depth API), posa del dispositivo, altri parametri della fotocamera per stabilire la mappatura del mondo fisico e degli oggetti virtuali nella scena.
- **Data structure:** i valori di profondità associati a ciascun pixel sono memorizzati all’interno di un’immagine prospettica (a bassa risoluzione).
Esistono 3 strutture dati differenti:
 - *array di profondità*: memorizza la profondità in un array 2D di un’immagine orizzontale con numeri interi a 16 bit. È possibile ricavare il valore della profondità di un qualsiasi punto dello schermo.
 - *mesh di profondità*: fornisce una ricostruzione 3D in tempo reale dell’ambiente circostante per ciascuna mappa di profondità. In figura (a) di 4.5 nella pagina seguente è rappresentata un’immagine di profondità che specifica il valore di profondità di ciascun pixel in base al colore (i pixel più luminosi indicano regioni più lontane). Ognuno di questi valori verrà proiettato nel template mesh tassellato generando una mesh di profondità in tempo reale (c) costituita dall’interconnessione di superfici triangolari.
 - *texture della profondità* è una texture che rappresenta la profondità della scena inquadrata dalla fotocamera.
- **Conversion Utilities and Algorithm** : l’utilizzo di algoritmi di conversione è diverso a seconda della funzionalità che viene offerta. Per gestire la fisica



Fonte: <https://augmentedperception.github.io/depthlab/assets>

Figura 4.5: Generazione di una depth mesh.

dell’ambiente, ombre, occlusione, mappatura di texture e molto altro sono necessari diversi approcci per l’elaborazione della profondità.

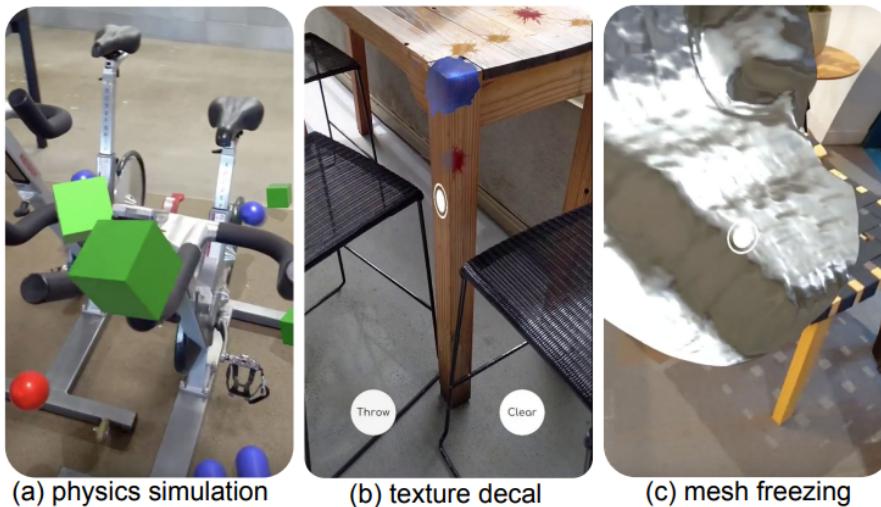
- **DepthLab Algorithms:** sulla base delle strutture dati gli algoritmi si dividono in 3 categorie:

- *profondità localizzata*: adatto per calcolare misurazioni fisiche, stimare vettori normali, muovere avatar virtuali in giochi AR (creare dei percorsi per evitare che l’avatar collida con qualche oggetto come nella figura 4.6 nella pagina successiva). Utilizza l’array di profondità per operare su un numero ridotto di punti.
- *profondità superficiale*: aggiorna mesh di profondità per gestire collisioni, texture decal (texture che vengono applicate su una superficie di un oggetto con una specifica proiezione), fisica dell’ambiente, riconoscimento geometrico delle ombre. (Esempi in figura 4.7 nella pagina seguente)
- *profondità densa*: utilizzato sul rendering di effetti sensibili alla profondità, reilluminazione, messa a fuoco, occlusione. (Esempi in figura 4.8 nella pagina successiva)



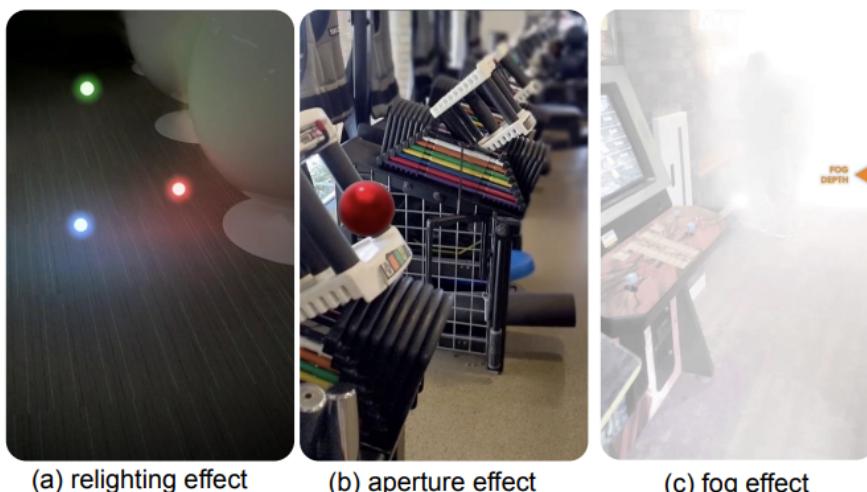
Fonte: <https://augmentedperception.github.io/depthlab/assets/>

Figura 4.6: Pianificazione di un percorso per un avatar virtuale.



Fonte: <https://augmentedperception.github.io/depthlab/assets/>

Figura 4.7: Esempi di profondità superficiale.



Fonte: <https://augmentedperception.github.io/depthlab/assets/>

Figura 4.8: Esempi di profondità densa.

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.

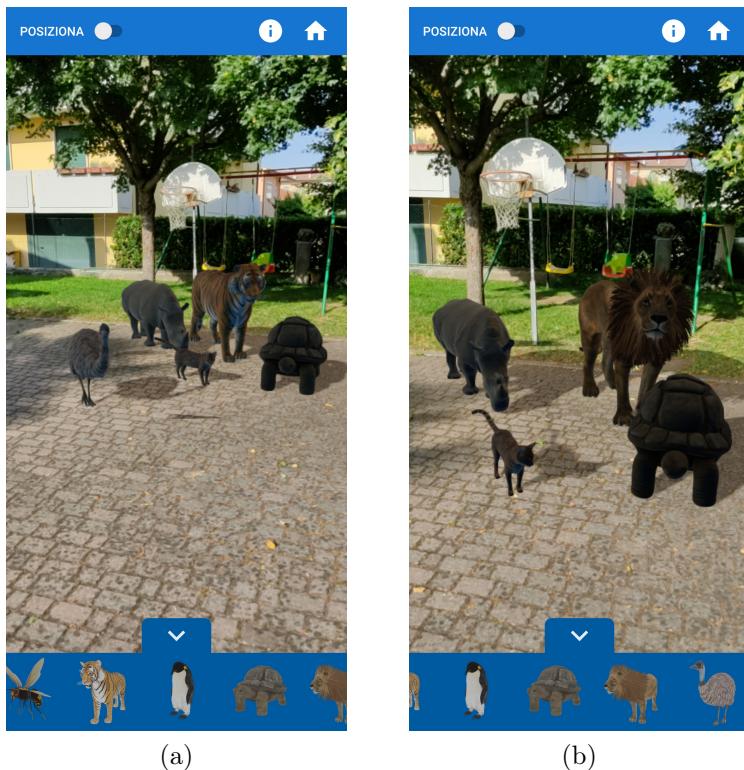


Figura 5.1: Esempio di molteplici hitTest su un piano in Plane Detection.

Un potenziale test per stimare le performance dell’interazione con l’utente potrebbe consistere nel posizionare un numero notevole di oggetti virtuali nella scena ed esaminare se le performance diminuiscono con l’incremento del numero di oggetti. Si potrebbe scoprire se esiste una soglia di numero di oggetti per la quale l’applicazione si arresta. Si consiglia di eliminare anchor inutili per evitare che le performance diminuiscano notevolmente.

6. Oriented Points

ARCore utilizza i punti orientati quando vengono toccate superfici che non sono piane (figura 6.1). Attorno al punto individuato dal tocco vengono esaminati dei punti caratteristici grazie ai quali è possibile stimare l'angolo dell'intersezione [15]. Il punto orientato è costituito dal risultato del hitTest che prende in 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 attorno al hitTest; 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 (unitario) 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.



Fonte: <https://developers.google.com/ar/develop/java/depth/developer-guide>

Figura 6.1: Esempio di punti orientati su una superficie arbitraria

7. HitTest

Un *HitTest* è il risultato che viene restituito quando viene toccato un determinato oggetto **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 8 a pagina 24 relativo ad Anchor e 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.

7.1 Definizione e gestione di un HitTest

E' possibile ricevere un HitTest di tipo diverso come descritto dal listing 7.1.

```

1 // I risultati dell'hit-test sono ordinati per distanza crescente dalla fotocamera.
2 val hitResultList =
3     if (usingInstantPlacement) {
4         // Se si usa la modalità Instant Placement, il valore in
5         // APPROXIMATE_DISTANCE_METERS determina quanto lontano sarà
6         // piazzato l'anchor, dal punto di vista della fotocamera.
7         frame.hitTestInstantPlacement(tap.x, tap.y, APPROXIMATE_DISTANCE_METERS)
8         // I risultati dell'Hit-test usando Instant Placement
9         // avranno un solo risultato di tipo InstantPlacementResult.
10    } else {
11        frame.hitTest(tap)
12    }
13
14 // Il primo hit result di solito è il più rilevante per rispondere agli input dell'utente.
15 val firstHitResult = hitResultList.firstOrNull { hit ->
16     val trackable = hit.trackable!!
17
18     if(trackable is DepthPoint){
19         // Sostituisci con un qualsiasi oggetto Trackable
20         true
21     } else {
22         false
23     }
24 }
25
26 if (firstHitResult != null) {
27     // Utilizza l'hit result. Ad esempio crea un anchor su tale punto di interesse.
28     val anchor = firstHitResult.createAnchor()
29     // Utilizzo dell'anchor...
30 }
```

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` sottostante su cui è avvenuto 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 25.

Oltre all'oggetto `hitTest` è stato molto importante `hitTestResult` definito nella documentazione di `sceneView`. Questo oggetto mantiene tutti gli `hitTest` che vengono creati quando l'utente tocca lo schermo. Inoltre, contiene le informazioni associate al nodo che è stato colpito dal `hitTest`. Quando l'utente posiziona un'animale in un piano, viene creato un oggetto `anchorNode` passando al costruttore l'anchor generato dal `hitTest`. Successivamente, viene aggiunto un oggetto `TransformableNode` come nodo figlio di `AnchorNode`. Questo tipo di nodo può essere utilizzato per aggiungere un oggetto `Node` come figlio, per eseguire operazioni di traslazione, selezione, rotazione e scala. L'utilizzo di `hitTestResult` è stato utile nell'eliminazione degli animali perché ci ha dato la possibilità di ricavare l'oggetto `Node` associato all'animale che l'utente voleva eliminare. Per rilevare un `hitTestResult` viene invocato il metodo `setOnTouchListener(delNode)` su `TransformableNode` dove `delNode` è un oggetto di tipo `Node.OnTouchListener`.

Nell'esempio 7.2 è riportato il codice dell'eliminazione di un nodo dalla scena.

```

1 //Listener per eliminare i nodi
2 val delNode = Node.OnTouchListener { hitTestResult , motionEvent ->
3     if(switchButton.isChecked){
4         // Prima chiamata ad ArFragment per gestire TrasformableNode
5         arFragment.onPeekTouch(hitTestResult , motionEvent)
6
7         // La rimozione si verifica con un evento ACTION UP
8         if (motionEvent.action == MotionEvent.ACTION_UP) {
9
10            if (hitTestResult.node != null && switchButton.isChecked) {
11
12                //Restituisce il nodo che è stato colpito dal hitTest
13                val hitNode: Node? = hitTestResult.node
14
15                hitNode!!.renderable = null
16
17                hitNode.parent = null
18
19                //Eliminazione di tutti i figli del nodo
20                val children = hitNode.children
21                if(children.isNotEmpty() && children != null){
22                    for (i in 0 until children.size){
23                        children[i].renderable = null
24                    }
25                }
26
27                arFragment.arSceneView.scene.removeChild(hitNode)
28            }
29        }
30    }
31
32 }
```

Listing 7.2: Eliminazione di un nodo dalla scena in Plane Detection

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 [17]. 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. Si veda il listing 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. Si veda il listing 8.2 nella pagina seguente.

```

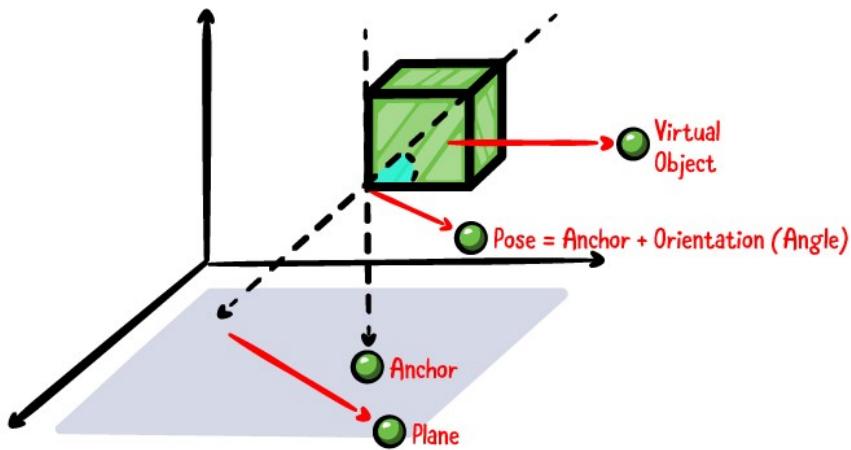
1 // Evento che si verifica quando viene toccato un piano
2 arFragment.setOnTapArPlaneListener { hitResult, plane, motionEvent ->
3
4     // Se siamo nella modalità place model
5     if (!switchButton.isChecked) {
6         arFragment.arSceneView.scene.addChild(AnchorNode(hitResult.createAnchor()))
7             .apply {
8
9                 // Crea il transformable model e lo aggiunge all'anchor
10                addChild(TransformableNode(arFragment.transformationSystem).apply {
11                    setModel()
12                    renderable = objRenderable
13                    //...
14                })
15            }
16    }
17 }
```

Listing 8.1: Definizione Anchor in Plane Detection.

```

1 // Per ogni immagine tracciata se non è presente il modello,
2 // esso viene immediatamente costruito e 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                 renderObject(
11                     arFragment,
12                     augmentedImage.createAnchor(augmentedImage.centerPose,
13                     namesobj[i]
14                 )
15             }
16             renderobj[i] = true
17         }
18     }
19 }
```

Listing 8.2: Definizione Anchor in Augmented Images.



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

Figura 8.1: Oggetto virtuale in un piano

La figura 8.1 mostra 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.

La figura 8.2 nella pagina seguente riporta degli esempi tratti dalla nostra applicazione in cui si può notare che il pinguino rimane nello stesso punto da qualsiasi prospettiva e distanza.

La figura 8.3 nella pagina successiva tratta dalla nostra applicazione mostra che nella modalità *Augmented Images* il pianeta rimane ancorato alla sua posizione.



Figura 8.2: Esempio di inquadrature differenti in Plane Detection

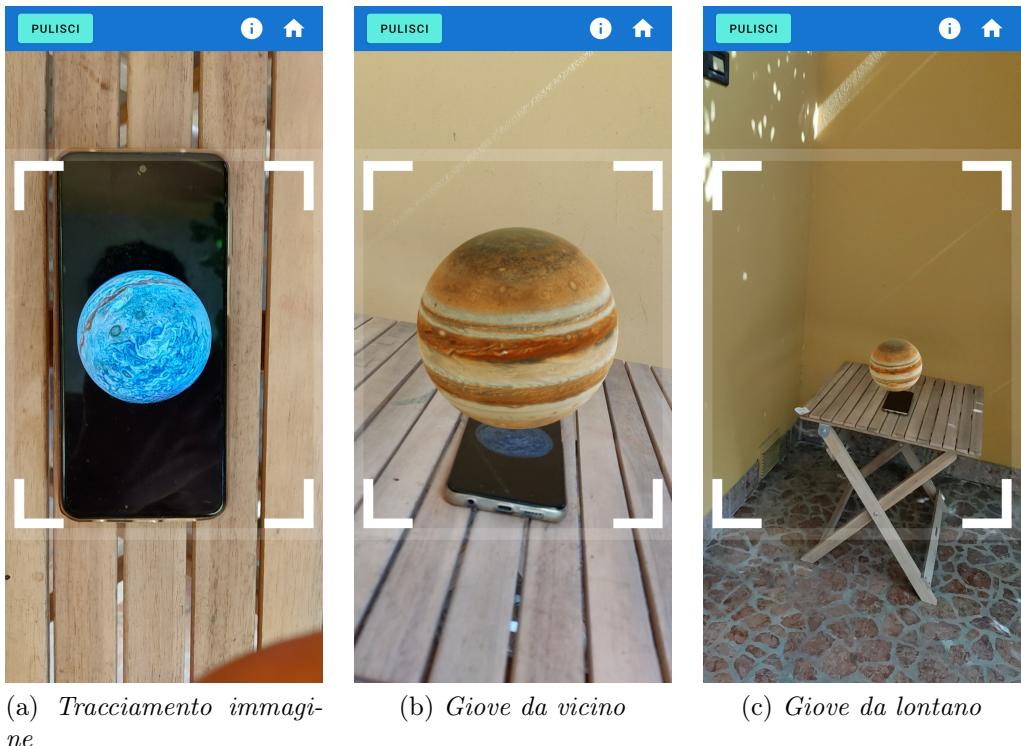


Figura 8.3: Esempio di rilevamento in Augmented Images

9. 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 [21]. Oltre ad essa, l'API fornisce un *center pose* e tre *region pose*, come descritti dalla figura 9.1 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 [9].

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.

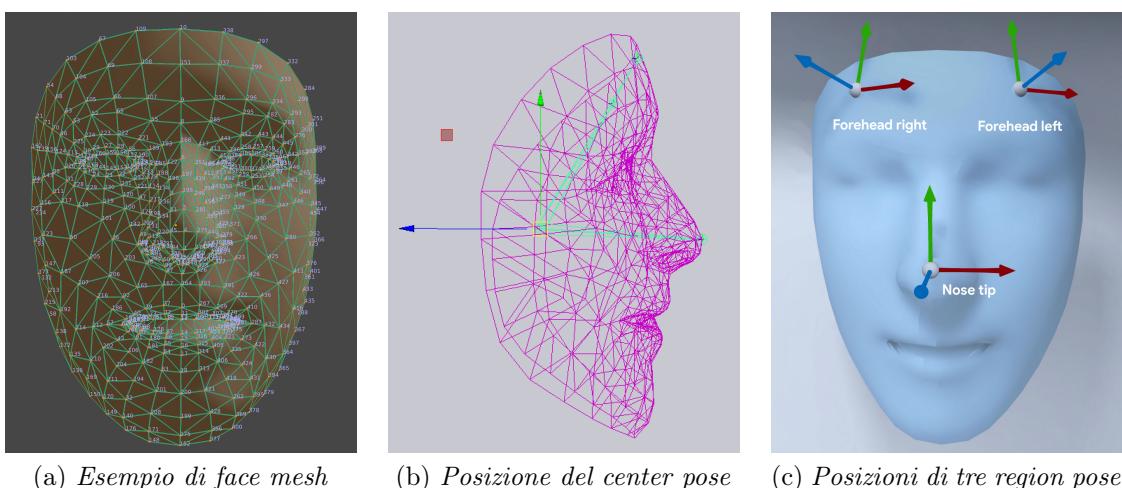


Figura 9.1: Elementi ottenuti tramite l'API Augmented Faces

9.1 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 9.1 tratto dalla guida ufficiale Google.

```

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 9.1: 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 9.2 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 9.2: Utilizzo della modalità Augmented Faces.

10. 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 [20].

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.

10.1 Funzionamento

La creazione e la diffusione dell'anchor avviene tramite connessione internet in quattro passaggi, descritti ad alto livello come segue. La figura 10.1 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.

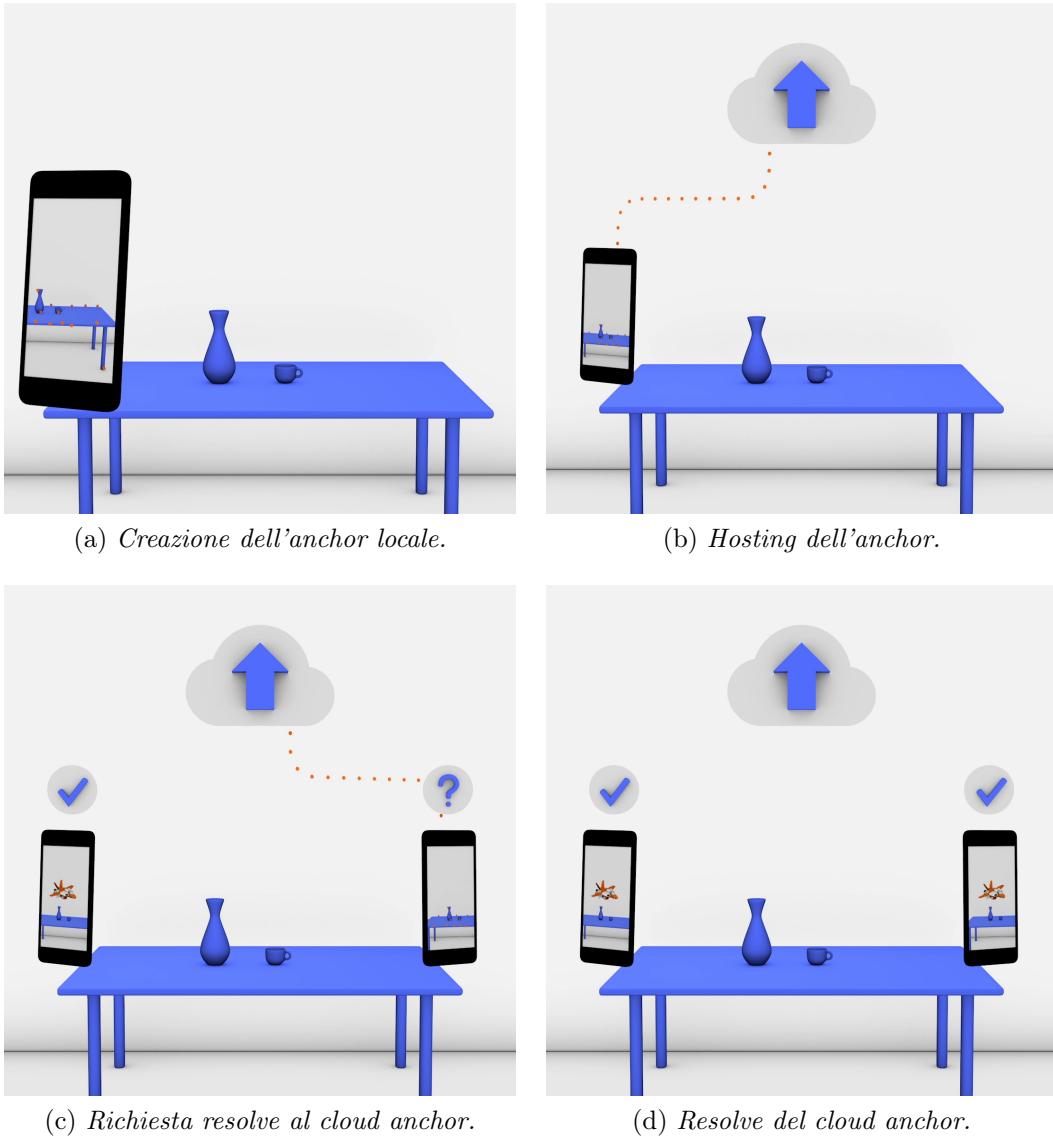


Figura 10.1: Funzionamento ad alto livello di Cloud Anchor API.

10.2 Configurazione 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 deve essere impostata per poter utilizzare l'API Cloud Anchors, come descritto dal listing 10.1 tratto dalla documentazione ufficiale.

```

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

```

Listing 10.1: Configurazione della modalità Cloud Anchor.

10.2.1 Autenticazione

Se il cloud anchor deve poter avere processi host/resolve di durata massima 24 ore, come ad esempio per giochi virtuali multigiocatore, è necessaria un'autenticazione tramite chiave dell'app. Tale autenticazione si compie generando una *API key* per il progetto cloud dal *Google Cloud Console* e aggiungendola nel campo `android:value` in un tag `meta-data`, nel campo `application` del file `AndroidManifest.xml` dell'applicazione, come spiegato dal listing 10.2.

```

1 <meta-data
2   android:name = "com.google.android.ar.API_KEY"
3   android:value = "API_KEY"/>

```

Listing 10.2: Autenticazione con API key.

Per cloud anchor che devono persistere per una durata compresa tra 1 e 365 giorni invece, è necessaria un'autenticazione tramite *OAuth client*, che associa l'applicazione android al progetto di Google Cloud Platform. Tale autenticazione richiede una chiave *SHA-1 fingerprint* che si può generare con il task `signingReport` di Gradle.

10.2.2 Hosting

Il metodo `hostCloudAnchorWithTtl(anchor:Anchor, ttlDays:Int)` della classe `Session` permette di iniziare l'hosting di `anchor` con una durata di `ttlDays` giorni, un numero positivo compreso tra 1 e 365; per hosting con durata fino a 24 ore è invece necessaria l'invocazione del metodo `hostCloudAnchor(anchor:Anchor)`. Entrambi i metodi restituiscono il nuovo anchor, con la stessa posizione di `anchor` e con stato `Anchor.CloudAnchorState.TASK_IN_PROGRESS`. Il listing 10.3 tratto dal codelab [25] realizza l'hosting tramite un oggetto della classe wrapper `CloudAnchorManager`.

```

1 val cloudAnchorManager : CloudAnchorManager = CloudAnchorManager()
2
3 // Realizza l'hosting per l'anchor currentAnchor con durata 300 giorni
4 cloudAnchorManager.hostCloudAnchor(session, currentAnchor, 300, this ::
    onHostedAnchorAvailable);

```

Listing 10.3: Hosting di un cloud anchor.

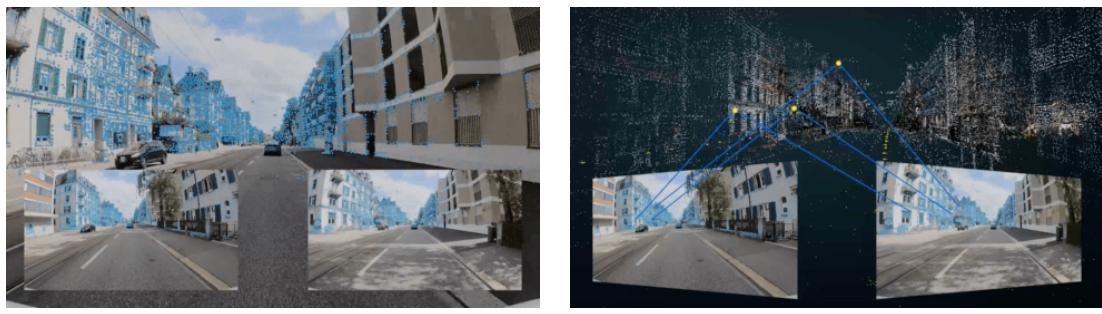
10.2.3 Resolving

Il metodo `resolveCloudAnchor(String cloudAnchorId)` della classe `Session` compie il resolving, creando un anchor sulla base delle informazioni 3D inviate e comparate con la mappa 3D caricata durante l'hosting, per poter posizionare correttamente l'oggetto rispetto al dispositivo che sta compiendo il resolving.

11. Geospatial API

L'API *Geospatial* è una funzionalità aggiunta a maggio 2022 al framework ARCore, che utilizza i dati di *Google Earth 3D* e *Google Maps Street View* per creare contenuti AR basati sulla posizione geografica. L'API sfrutta il *global localization*, il quale combina il *VPS - Visual Positioning Service*, un servizio Google che analizza l'ambiente circostante attraverso la fotocamera per determinare la posizione, *Street View*, che fornisce un database di immagini di luoghi, e il machine learning per migliorare la determinazione della posizione del dispositivo [23].

Il Geospatial API compara le informazioni provenienti dalla fotocamera (figura 11.1a) e dai sensori del dispositivo, come il GPS, con miliardi di immagini 3D estratte tramite machine learning da Street View (figura 11.1b) per determinare la posizione e l'orientamento del dispositivo, per poi mostrare contenuti AR posizionati correttamente rispetto all'utente, come spiegato in [12].



(a) Immagini della fotocamera del dispositivo (b) Confronto con immagini Street View

Figura 11.1: Ricostruzione delle funzionalità di Geospatial API.

11.1 Configurazione e utilizzo

La sessione ARCore deve abilitare l'utilizzo di Geospatial API, come descritto dal listing 11.1 tratto dalla documentazione ufficiale [7].

```
1 // Abilita il Geospatial API.  
2 session.configure(session.config.apply{ geospatialMode= Config.GeospatialMode.  
    ENABLED })
```

Listing 11.1: Configurazione della modalità Geospatial API.

11.1.1 Configurazione di VPS

L'utilizzo di Visual Positioning System impone che l'app sia associata a un progetto Google Cloud Project con abilitato il ARCore API. È inoltre necessaria un'autenticazione tramite *OAuth client*, oppure con *API key*. Si veda il paragrafo 10.2.1 a pagina 32 del capitolo Cloud Anchor per specifiche su entrambi i tipi di autenticazione.

Sono necessari inoltre i permessi per accedere alla posizione e ad internet per comunicare con il servizio online Geospatial API, da dichiarare nel campo `manifest` del file `AndroidManifest.xml`, come descritto dal listing 11.2.

```

1 <manifest ...>
2   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
3   <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
4   <uses-permission android:name="android.permission.INTERNET"/>
5 </manifest>
```

Listing 11.2: Richiesta di permessi per l'uso di Geospatial API.

11.1.2 Calcolo della posizione

La posizione può essere presa da un oggetto della classe `Earth`, ricevuto dalla sessione ARCore `session`, come descritto dal listing 11.3. Se l'oggetto di tipo `Earth` ha stato `TrackingState.TRACKING`, la posizione può essere ottenuta tramite un oggetto di tipo `GeospatialPose`, che contiene la latitudine e la longitudine, l'altitudine e un'approssimazione della direzione verso cui il dispositivo è rivolto.

```

1 val earth = session.earth
2 // Verifico che sia in stato TRACKING
3 if (earth?.trackingState == TrackingState.TRACKING) {
4   val cameraGeospatialPose : GeospatialPose = earth.cameraGeospatialPose
5
6   // Salvataggio della latitudine in gradi
7   val latitude : Double = cameraGeospatialPose.latitude
8   // Salvataggio della longitudine in gradi
9   val longitude : Double = cameraGeospatialPose.longitude
10  // Salvataggio dell'altitudine in metri
11  val elevation : Double = cameraGeospatialPose.altitude
12  // Salvataggio dell'orientamento in gradi
13  val heading : Double = cameraGeospatialPose.heading
14}
```

Listing 11.3: Calcolo della posizione corrente.

L'oggetto `GeospatialPose` specifica anche l'accuratezza *A* dei dati ricevuti, tramite i metodi `getHeadingAccuracy()`, `getHorizontalAccuracy()` e `getVerticalAccuracy()`. I valori restituiti dai metodi hanno stessa unità di misura dei valori stimati *E* di cui specificano la precisione, cioè gradi per la latitudine, la longitudine e l'orientamento

e metri per l'altitudine, e specificano che la posizione reale R è compresa con una probabilità del 68% nell'intervallo:

$$R \in [E - A, E + A].$$

Ad esempio, se il metodo `GeospatialPose.getHeading()` restituisce il valore $E = 60^\circ$ e il metodo `GeospatialPose.getHeadingAccuracy()` ritorna una precisione di $A = 10^\circ$, il valore reale sarà con probabilità del 68% nell'intervallo $R \in [50^\circ, 70^\circ]$. Un valore alto di accuratezza quindi garantisce una precisione minore.

11.1.3 Posizionamento di un anchor Geospatial

Per il posizionamento di un anchor, i valori di latitudine e longitudine devono essere dati rispettando le specifiche WGS84, mentre l'altitudine è definita come la distanza in metri dall'elissoide definito dallo stesso standard. L'orientamento dell'anchor invece viene fatto con l'utilizzo di un quaternione (qx, qy, qz, qw). Si veda il listing 11.4 per un esempio di creazione dell'anchor.

```

1 if (earth.trackingState == TrackingState.TRACKING) {
2     val anchor = earth.createAnchor (
3         /* Valori della posizione */
4         latitude, longitude, altitude,
5         /* Valori della rotazione */
6         qx, qy, qz, qw)
7         // ...
8 }
```

Listing 11.4: Posizionamento di anchor Geospatial.

L'altitudine dell'anchor, se esso viene posizionato vicino all'utente, può avere lo stesso valore dell'altitudine restituita dal metodo `GeospatialPose.getAltitude()`. Se invece l'anchor deve avere un'altitudine diversa da quella dell'utente, essa può essere ricavata dall'API Google Maps, forzando la prospettiva 2D e convertendo il valore restituito, basato sullo standard EGM96, nella codifica WGS84.

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. *Geospatial developer guide for Android (Kotlin/Java)*. Giu. 2022. URL: <https://developers.google.com/ar/develop/java/geospatial/developer-guide>.
- [8] Google developers. *Get the lighting right*. Mag. 2022. URL: <https://developers.google.com/ar/develop/lighting-estimation>.
- [9] 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>.
- [10] Google developers. *Use Depth in your Android app*. Mag. 2022. URL: <https://developers.google.com/ar/develop/java/depth/developer-guide>.

-
- [11] Google developers. *Use Raw Depth in your Android app.* Mag. 2022. URL: <https://developers.google.com/ar/develop/java/depth/raw-depth>.
 - [12] Google developers. *Using Global Localization to Improve Navigation.* Mag. 2022. URL: <https://developers.googleblog.com/2022/05/Make-the-world-your-canvas-ARCore-Geospatial-API.html>.
 - [13] Math works developers. *Che cos'è la SLAM?* 2022. URL: <https://it.mathworks.com/discovery/slam.html>.
 - [14] Ruofei Du et al. «Experiencing Real-Time 3D Interaction with Depth Maps for Mobile Augmented Reality in DepthLab». In: *Adjunct Publication of the 33rd Annual ACM Symposium on User Interface Software and Technology*. UIST '20 Adjunct. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 108–110. ISBN: 9781450375153. DOI: 10.1145/3379350.3416136.
 - [15] Anna Eklind e Love Stark. *An exploratory research of ARCore's feature detection.* 2018.
 - [16] 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.
 - [17] Hasret Gumgumcu. «Evaluation Framework for Proprietary SLAM Systems exemplified on Google ARCore». B.S. thesis. Giu. 2019.
 - [18] Anders Henrysson, Mark Billinghurst e Mark Ollila. «AR Tennis». In: SIGGRAPH '06 (2006), 1–es. DOI: 10.1145/1179133.1179135.
 - [19] Andreas Jakl. *Basics of AR: SLAM – Simultaneous Localization and Mapping.* Ago. 2018. URL: <https://www.andreasjakl.com/basics-of-ar-slam-simultaneous-localization-and-mapping/>.
 - [20] 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>.
 - [21] Zainab Oufqir, Abdellatif El Abderrahmani e Khalid Satori. «ARKit and ARCore in service to augmented reality». In: *2020 International Conference on Intelligent Systems and Computer Vision (ISCV)*. 2020, pp. 1–7. DOI: 10.1109/ISCV49265.2020.9204243.
 - [22] Tommy Palladino. *Google Adds Raw Depth API to Improve Spatial Awareness & Depth Data for Android AR Apps.* Mag. 2021. URL: <https://mobile-ar.reality.news/news/google-adds-new-depth-api-improve-spatial-awareness-depth-data-for-android-ar-apps-0384659/>.

- [23] Tilman Reinhardt. *Using Global Localization to Improve Navigation*. Feb. 2019. URL: <https://ai.googleblog.com/2019/02/using-global-localization-to-improve.html>.
- [24] 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*. UI-ST '95. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1995, pp. 29–36. ISBN: 089791709X. DOI: 10.1145/215585.215639.
- [25] Fred Sauer e Dereck Bridie. *ARCore Cloud Anchors with persistent Cloud Anchors*. Ott. 2021. URL: <https://codelabs.developers.google.com/codelabs/arcore-cloud-anchors>.
- [26] Aleksi Suonsivu. «RGBD SLAM Based 3D Object Reconstruction and Tracking: Using Google ARCore». B.S. thesis. 2020.
- [27] 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.
- [28] 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.