1. 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.

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. (Esempio **??** a pagina ??)

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

Listing 1.1: 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       // Retrieve the depth image for te current frame, if available
4       try{
5        frame.acquireDepthImage16Bits().use{ depthImage ->
6         //Use the depth image here
7        }
8       } catch(e: NotYetAvailableException){
9        // This means that depth data is not available yet.
10         // Depth data will not be available if there are no tracked
11         // feature points.  This can happen when there is no motion, or when the
12         // camera loses its ability to track objects in the surrounding
13         // environment.
14      }
```

Listing 1.2: Controllo supporto depth API

1.2 Depth Hit Test

Hit Test sono 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 id Plane || trackable is Point || trackable is DepthPoint){
9         val anchor=hit.createAnchor()
10        //Use anchor here
11       }
12      }
```

Listing 1.3: Depth Hit Test