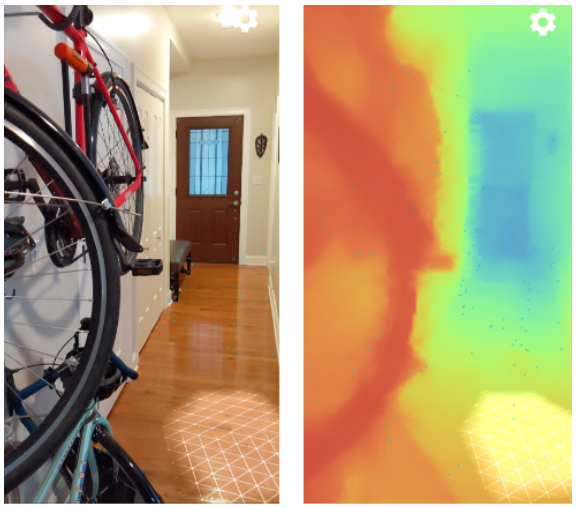
1. Depth understanding

1.1 API Depth

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 1.1 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; dunque è necessario attivarla manualmente quando viene creata una 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 1.1: Esempio di mappa di profondità

1.1.1 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. (Codice 1.1)

```

    val config = session.config

    // Check whether the user's device supports the Depth API.
    val isDepthSupported = session.isDepthModeSupported(Config.DepthMode.AUTOMATIC)
    if (isDepthSupported) {
        config.depthMode = Config.DepthMode.AUTOMATIC
    }
    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()**.(Codice 1.2)

```
    val frame = arFragment.arSceneView.frame

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

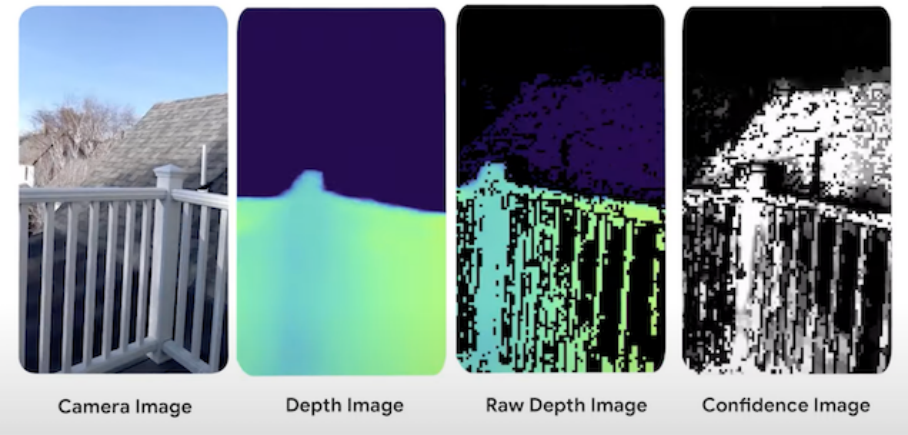
Listing 1.2: Estrazione di un'immagine profonda

```kotlin
/** Obtain the depth in millimeters for [depthImage] at coordinates ([x], [y]). */
fun getMillimetersDepth(depthImage: Image, x: Int, y: Int): Int {

    // The depth image has a single plane, which stores depth for each
    // pixel as 16-bit unsigned integers.
    val plane = depthImage.planes[0]

    //The distance between adjacent pixel samples, in bytes
    val pixelStride= x * plane.pixelStride

    //The row stride for this color plane, in bytes.
    val rowStride= y * plane.rowStride

    val byteIndex = pixelStride + rowStride

    //Retrieves this buffer's byte order.
    val buffer = plane.buffer.order(ByteOrder.nativeOrder())

    //Reads two bytes at the given index, composing them into a short value
    // according to the current byte order.
    val depthSample = buffer.getShort(byteIndex)

    return depthSample.toInt()
}
```

Listing 1.3: Estrazione di informazioni da un'immagine profonda

1.2 Raw Depth API

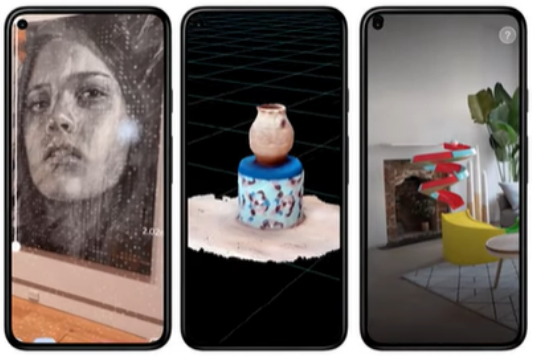
Le ARCore Raw Depth API forniscono informazioni più precise sulla profondità di alcuni pixel di un'immagine e permettono di rappresentare la geometria della scena generando delle **immagini di profondità grezze**. A queste immagini vengono associate delle **immagini di affidabilità** che stabiliscono il grado di confidenza di ciascun pixel (associato ad un valore di profondità). 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 corrispondente pari a 0.



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

Figura 1.2: Differenze tra immagini in API Raw Depth

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).



Fonte: https://www.youtube.com/watch?v=13WugTMOdSs

Figura 1.3: Applicazioni Raw Depth API

1.2.1 Sessione ARCore con Raw depth API

Inizialmente si deve effettuare il controllo sulla compatibilità del dispositivo come riportato nel codice **??** a pagina ??.
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. Codice riportato **??** a pagina ??

```
    try {

     //First of all, retrieve raw depth image
      // Raw Depth image is in uint16, at GPU aspect ratio, in native orientation.
      frame.acquireRawDepthImage16Bits().use { rawDepth ->

        // Confidence image is in uint8, matching the depth image size.
        frame.acquireRawDepthConfidenceImage().use { rawDepthConfidence ->

            // Compare timestamps to determine whether depth is is based on new
            // depth data, or is a reprojection based on device movement.
            val thisFrameHasNewDepthData = frame.timestamp == rawDepth.timestamp

            if (thisFrameHasNewDepthData) {

              val depthData = rawDepth.planes[0].buffer
              val confidenceData = rawDepthConfidence.planes[0].buffer
              val width = rawDepth.width
              val height = rawDepth.height
              someReconstructionPipeline.integrateNewImage(
                depthData,
                confidenceData,
                width = width,
                height = height
              )
            }
        }
      }
    } catch (e: NotYetAvailableException) {
      // Depth image is not (yet) available.
    }
```

Listing 1.4: Estrazione di un'immagine di confidenza

1.3 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. (Codice in 1.5)

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

Listing 1.5: Depth Hit Test