Chapter 1

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 **??** 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.

1.1 Definizione e gestione di un HitTest

E' possibile ricevere un `HitTest` di tipo diverso come descritto dal listing **??**.

[caption=Filtraggio hitTest in base al tipo., label=lst: hitTest-filter, language=Kotlin] // I risultati dell'hit-test sono ordinati per distanza crescente dalla fotocamera. val hitResultList = if (usingInstantPlacement) // Se si usa la modalità Instant Placement, il valore in // APPROXIMATE DISTANCE METERS determina quanto lontano sarà // piazzato l'anchor, dal punto di vista della fotocamera. frame.hitTestInstantPlacement(tap.x, tap.y, APPROXIMATE$_{D}ISTANCE_{M}ETERS$)//$Irisultatidell'Hit - testusandoInstantPlacement//avrannounsolorisultatoditipoInstantPlacementResult.elseframe.hi$ // Il primo hit result di solito è il più rilevante per rispondere agli input dell'utente. val firstHitResult = hitResultList.firstOrNull hit -¿ val trackable = hit.trackable!!
if(trackable is DepthPoint) // Sostituisci con un qualsiasi oggetto Trackable true else false
if (firstHitResult != null) // Utilizza l'hit result. Ad esempio crea un anchor su tale punto di interesse. val anchor = firstHitResult.createAnchor() // Utilizzo dell'anchor...

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 **??**.
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 perchè 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 setOnTouch-Listener(delNode) su TransformableNode dove delNode è un oggetto di tipo Node.OnTouchListener.
Nell'esempio **??** è riportato il codice dell'eliminazione di un nodo dalla scena.

[caption=Eliminazione di un nodo dalla scena in Plane Detection, label=lst: delete-node, language=Kotlin] //Listener per eliminare i nodi val delNode = Node.OnTouchListener hitTestResult, motionEvent -¿ if(switchButton.isChecked) Log.d(TAG, "handleOnTouch") // Prima chiamata ad ArFragment per gestire TrasformableNode arFragment.onPeekTouch(hitTestResult, motionEvent) //La rimozione si verifica con un evento ACTION$_U$ $Pif(motionEvent.action == MotionEvent.ACTION_UP)$ if (hitTestResult.node != null switchButton.isChecked) Log.d(TAG, "handleOnTouch hitTestResult.getNode() != null") //Restituisce il nodo che è stato colpito dal hitTest val hitNode: Node? = hitTestResult.node hitNode!!.renderable = null hitNode.parent = null //Eliminazione di tutti i figli del nodo val children =hitNode.children if(children.isNotEmpty() children != null) for (i in 0 until children.size) children[i].renderable = null arFragment.arSceneView.scene.removeChild(hitNode) true