This tutorial shows how to write an AR View that selects the scene that the camera is pointing at.

1) Implementing the AR View Activity

In order to create the point-to-select functionality we must make some changes to the standard AR View Activity. We define a field of type RelativeLayout on the class. We will call this field `uiLayout`, and it will hold the root element of the layout corresponding to the AR View Activity.

Another field we need is, `reticleImage`, of type ImageView, which will hold the image that is displayed for the targeting reticle of the AR View HUD. This field will point to one of two different drawables, depending on whether or not the camera is pointing at a scene object or not.

We will also declare a field of type OAScene, called `lastSceneSelected`, which will store a reference to the last scene object that was selected, in order to avoid selecting the same object successively when the target reticle is left on it. We will initialise this field to null.

We need two ints, `screenWidth` and `screenHeight`, which will store the dimensions of the device screen in order to determine the coordinates of the centre of the screen where we will trigger artificial touch events.

Finally we must declare a field to store the timer that will determine when to trigger our artificial touch events. We will declare a field `timer`, of type Timer, and initialise it to null.

Next we override the `onResume()` method in order to start the timer every time the activity is resumed:

```java
@Override
protected void onResume() {
```
super.onResume();

// start timer tasks
startTimer();

We can also override onPause() to stop the timer if we intend to have other activities opening from the AR View.

Our setupUILayout() method must retrieve the screen dimensions, and set the value of reticleImage. It will look like this:

```java
@Override
protected void setupUILayout(View arView) {
    super.setupUILayout(arView);
    DisplayMetrics displaymetrics = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(displaymetrics);
    screenHeight = displaymetrics.heightPixels;
    screenWidth = displaymetrics.widthPixels;
    // add main ui layout
    LayoutInflater controlInflater = LayoutInflater.from(getBaseContext());
    uiLayout = (RelativeLayout)controlInflater.inflate(R.layout.arview, null);
    addContentView(uiLayout, new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
    // setup reticle for point-to-select
    reticleImage = (ImageView)findViewById(R.id.reticleImage);
}
```

Since we want scenes to be unselected when the reticle is not focused on a scene object, we need to override the onTouchedScene() method rather than the onSceneSelected() method. This is because onTouchedScene() is called even when a touch event occurs on an area where no scene object is present, in which case it is passed a null parameter. Our implementation of onTouchedScene() should look like this:

```java
@Override
protected void onTouchedScene(OAScene scene) {
    if (scene != null) {
        // scene selected
        // show reticle enabled
        reticleImage.setVisibility(View.VISIBLE);
        reticleImage.setImageResource(R.drawable.reticle_blue);
        if (scene != lastSceneSelected) {
            // new scene selected
            // set new scene and make toast
        }
    }
}
```
lastSceneSelected = scene;
Toast.makeText(this, "Scene: " + scene.getName(), Toast.LENGTH_SHORT).show();

} else if (scene == null) {
    // no scene selected
    // show reticle disabled
    reticleImage.setVisibility(View.VISIBLE);
    reticleImage.setImageResource(R.drawable.reticle_grey);
    // reset last scene variable
    lastSceneSelected = null;
}

This method first checks to see whether a scene object has been selected, and if so it sets the reticle image to blue to show that a scene is being pointed at. If the scene object that has been selected is different from the last selected object then the value of lastSceneSelected is set to the new scene and a toast with the name of the scene is displayed. If no scene is selected then the reticle is set to grey to show that no scene is being pointed at, and the value of lastSceneSelected is reset to null.

Now we must set up the timer. To do this we will define several private methods, startTimer(), stopTimer(), and timerTask(). startTimer() will look like this:

private void startTimer() {
    if (timer != null) {
        stopTimer();
    }
    timer = new Timer();
    timer.schedule(
        new TimerTask() {
            @Override
            public void run() {
                timerTask();
            }
        }, 500, 500);
}

This method creates the timer and schedules it to create touch events every 0.5 seconds. The actual touch events themselves are created in the timerTask() method.

stopTimer() is used to clear the current instance of the timer, it looks like this:

private void stopTimer() {

if (timer != null) {
    timer.cancel();
    timer.purge();
    timer = null;
}

} } Finally, timerTask() creates the touch events located at the centre of the screen.

private void timerTask() {
    getSceneView().triggerTouch( screenWidth / 2, screenHeight / 2 );
}

2) The AR View Layout

The layout required for this AR View is quite simple, it only requires a RelativeLayout as its root element in order to allow the targeting reticle to be positioned at the center of the layout, and it requires the ImageView that represents the targeting reticle. The ImageView layout element can be specified as follows:

<ImageView android:id="@+id/reticleImage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/reticle_grey"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

Where drawable/reticle_grey is the location of the targeting reticle drawable.

Now we have a working point-to-select interface in our AR View Activity.

If you have further problems or questions, visit our website (http://www.hitlabnz.org/mobileAR) and post your problem on the Mobile AR Framework Support Forum.