In this tutorial, we will see how to make simple animations with 3D models, and add them to the AR application. We assume that you have already build simple AR applications referring to the basic tutorial.

1.1 Setup the AR view
First of all, we create a SampleAnimationActivity which extends OAARComponentBase class.

```java
public class ARViewActivity extends OAARComponentBase {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // setup fake location
        Location sLoc = new Location("";
        sLoc.setLatitude(-43.522189);
        sLoc.setLongitude(172.58292);
        sLoc.setAltitude(0);
        sLoc.setAccuracy(0.1f);
        sLoc.setTime(System.currentTimeMillis());
        getSensorManager().addGPSFilter(new GPSFilterSpoofed(sLoc));
    }
}
```

1.2 Setup the scenes
Now we will edit the setupScenes() callback function and add the animated objects into the scene. We will later see how these animations are created.

```java
@Override
protected void setupScenes(OADataManager dataManager) {
    Create an instance of AnimatedObject class. This class is defined later in this tutorial.
    First we insert an AnimatedObject with model id as “tbox”. This is the name of the 3D model folder inside assets/OutdoorAR/3Dmodels location.

    OAScene scene1 = new AnimatedObject(
        -43.523177, 172.582512, "AnimatedObject1", "tbox");
```
AnimatedObject mObject1 = (AnimatedObject) scene1;
mObject1.animateRotation(2);
dataManager.addScene(scene1);

OAScene scene2 = new AnimatedObject
(-43.521137,172.583417, "AnimatedObject2", "15");
AnimatedObject mObject2 = (AnimatedObject) scene2;
mObject2.animateBounce();
dataManager.addScene(scene2);

super.setupScenes(dataManager);

2.1 Create the animation class
Next we will create a new class which will handle the animation of the models. Let us call this class as AnimatedObject. This class should be a subclass of OAScene.

public class AnimatedObject extends OAScene {

We will have to define some variables related to the animation that we are going to use.

    private int frameRate = 10; //frames per second
    private int duration = 1; // second
    private AtomicReference<Float> speed = new AtomicReference<Float>(0.2f);;
    private float tailLength = 40.0f; // length of the animated object
    private String modelid = null;

The constructor for this class will instantiate the OAScene variables such as latitude, longitude, name of the model and the model id.

    public AnimatedObject(double latitude, double longitude, String name, String modelId) {
        super(latitude, longitude, name);
        this.modelid = modelId;
    }

Here model id referes to the name of the folder in assets/OutdoorAR/3Dmodels/ location. So, if your model folder is named “box”, then the .obj and .mtl files would be present inside assets/OutdoorAR/3Dmodels/box location.

2.2 Create a bouncing animation
Next step is to define our animation. So, in this tutorial, we show how we can translate and rotate objects. First we define a method which will bounce a model using translations.

    public void animateBounce() {

AnimatedTransform animation = new AnimatedTransform();

We have to define the translations for each of the key frames. Here the number of key frames can be obtained by frameRate * duration.

    for (int i = 0; i < frameRate * duration - 1; i++) {
        Transform transform = new Transform();
        Translate the object along y axis.
        Vector3f translation = new Vector3f(0, i, 0);
        transform.setTranslation(translation);
        Add the transformation to the animation.
        animation.addTransform(transform, 1000 / frameRate * i);
    }

We will get the first frame in the key frames and add it to the end of the animation to complete an animation loop.

    animation.addTransform(
        animation.getAnimation().getKeyFrames().get(0).getKeyFrameObject(),
        1000 / frameRate * frameRate * duration);

We will set the animation speed here. The variable “speed” is earlier defined as a data member of the class.

    animation.getAnimation().setAnimationSpeed(speed);

Finally, we create an object of ModelData and assign the model id.

    ModelData model = new ModelData();
    model.addAnimation(animation);
    model.id = modelid;
    this.models.add(model);
}

We are now done with one animation - animateBounce().

2.3 Create a rotation animation
Now we define another method to rotate an object. We also make use of the idea of segments. In our example, we will create a rotating box. Not just one, but with 2 segments. So, here we will need to assign the transformation to each of these segments.
The following method would implement animationRotation.

```java
public void animateRotation(int segments) {
    AnimatedTransform[] animations = new AnimatedTransform[segments];
    ModelData[] models = new ModelData[segments];

    Define the axis along which we are rotating the object.
    Vector3f rotation = new Vector3f(0, 1, 0);

    The animation loop is defined below.
    for(int i = 0; i < frameRate * duration - 1; i++){
        for(int j = 0; j < segments; j++){
            Transform transform = new Transform();
            Translate the segments apart by some distance. Here we define it with respect to the tail length and number of segments.
            Vector3f translation = new Vector3f(0, 0, tailLength / segments * j);
            transform.setTranslation(translation);

            Define the rotation angle. We assign it as multiples of 10 degrees with each keyframe rotating the object by 0, 10, 20 degrees etc...
            float rotationAngle = -(float)Math.toDegrees(10.0f * i);
            transform.setRotation(rotation);
            transform.setRotationAngle(rotationAngle);

            Now, for each segment, we set the animation and the speed of the animation.
            if(animations[j] == null){
                animations[j] = new AnimatedTransform();
                animations[j].getAnimation().setAnimationSpeed(speed);
            }

            And then we add the transformation to the animation.
            animations[j].addTransform(transform, 1000/frameRate * i);
        }
    }
}
```
Next we generate the body segments using a for loop. So, we create ‘k’ models equal to the number of segments.

```java
for(int k = 0; k < segments; k++){
    if(models[k] == null){
        models[k] = new ModelData();
    }
}
```

Similar to how we did in the earlier example, we get the first frame in the key frames and add it to the end of the animation to complete an animation loop.

```java
animations[k].addTransform(animations[k].getAnimation().getKeyFrames().get(0).getKeyFrameObject(), 1000/frameRate * frameRate * duration);
```

And then add the animation to each of the segments.

```java
models[k].addAnimation(animations[k]);
```

Define the model id to be used for each of these segments. Here, we use the same model to form the different segments.

```java
models[k].id = modelid;
```

And finally, we add the model to the scenes.

```java
this.models.add(models[k]);
}
```

That is it. We have made a simple animation application which will rotate 2 segmented boxes and bounce a ball.
Screenshot: Rotating cubes