Tutoriel Réalité Augmentée avec Vuforia 6

Scénario

Il s'agit d'aider à monter une pièce en Légo. Il y a des pièces de Légo dans des sachets. Les indications de montage peuvent se faire en reconnaissant les pièces. Ici il est plus facile (et possible) de reconnaitre un « marker » posé sur un sachet.

Il s'agit ici de créer une application (basique) qui affichera les étapes concernées par un sachet.

Avertissement : si votre sdk cible est 23, vous aurez un problème de permissions. Le plus simple est de cibler un sdk précédent.

Démarrage du projet android

Il faut créer un projet en choisissant « empty activity » en ne cochant pas « backward compatibility » (On a une Activity, un peu plus simple).

Recopiez (ou créez) un dossier libs dans le dossier « app ». Ce dossier libs contient les deux jars « armeabi.jar » et « Vuforia.jar », disponibles sur le site du cours.

Dans le layout, à la place du TextView, mettez une progress bar (avec id)

```
<ProgressBar
    style="@android:style/Widget.ProgressBar"
    android:id="@+id/loading_indicator"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true" />
```

Dans le manifest, il faut mettre les permissions (opengl, caméra, internet, accès aux fichiers)

```
<uses-feature android:glEsVersion="0x00020000" />
<uses-feature android:name="android.hardware.camera" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Il faut aussi récupérer des classes de l'exemple : une classe SampleApplicationSession qui encapsule les appels à Vuforia, notamment en utilisant des AsyncTask pour l'initialisation, le chargement des éléments à reconnaitre (Tracker). Ces classes sont donc :

- La version utilisée pour les exemples est « vuforia-samples-core-android-6-0-112.zip ».
  La dernière version est disponible https://developer.vuforia.com/downloads/samples
- Dans le package com.vuforia.samples.SampleApplication , il y a donc la classe SampleApplicationSession , accompagnée de la classe SampleApplicationControl (Interface Logicielle pour la liaison entre votre application et Vuforia via la SampleApplicationSession) et SampleApplicationException
  SampleApplicationSession utilise des chaines de caractères R.string, il faudra donc également récupérer les valeurs correspondantes est les mettre dans le fichier strings.xml (dans res/values)

- Dans le package com.vuforia.samples.SampleApplication.utils, il faut des classes qui facilitent les manipulations openGL : CubeSharders, SampleApplicationGLVIew, SampleUtils, Texture

Vous allez maintenant faire, étape par étape, l'application.

1re étape : initialisation de Vuforia

Pour lancer Vuforia, la reconnaissance via la caméra et l'affichage openGL par-dessus l'image de la caméra, vous allez donc utiliser une SampleApplicationSession qui retournera les résultats de Vuforia via une Interface Logicielle SampleApplicationControl (et via aussi un Renderer qui sera fait plus tard). Le plus direct est que votre Activity implémente l'interface SampleApplicationControl. Pour que l'initialisation aille à son terme, il faut que les méthodes doInitTrackers et doLoadTackersData retourne « true » même si pour l'instant ces méthodes ne font rien (sinon, l'initialisation s'arrête).

Au lancement de l'application (dans le onCreate), il faut donc créer une SampleApplicationSession. Ensuite, il faut charger les textures qui seront utiles ultérieurement. Pour cela, vous utilisez un vecteur de Texture (la classe récupérée de l'exemple) et vous ajouter dedans lesdites textures. Les images doivent être dans un dossier « assets » qui est dans « app/src/main » :

```
mTextures = new Vector<Texture>();
mTextures.add(Texture.loadTextureFromApk("etape01.jpg", getAssets()));
// etc.
```

Finalement, pour lancer l'utilisation de Vuforia (cela ne fera encore rien), il faut initialiser via la méthode initAR :

```
vuforiaAppSession.initAR(this, ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

Une fois l'initialisation finie, la méthode onInitARDone (méthode de votre application, implémentation de l'interface logicielle SampleApplicationControl) sera appelée. C'est ici qu'il faudra lancer l'openGL, créer un renderer et lancer la reconnaissance (qui ne fera rien tant que doInitTrackers et doLoadTackersData ne seront pas complétées, ce qui sera fait plus tard).

2e étape : initialisation de la scène openGL

Un renderer est une classe qui implémente GLSurfaceView.Renderer et qui sera associé à la scène openGL pour afficher des éléments par-dessus la vidéo. Un renderer contient des méthodes pour l'initialisation d'openGL et une méthode onDrawFrame pour le dessin. Vous disposez d'une classe ARRender (à renommer en enlevant le .initial) à ajouter à votre application. Ce renderer ne fait qu'afficher le flux de la caméra (en préparant la suite…).

Dans onInitARDone, vous mettrez donc un code semblable à celui-là :

```
// en cas de problème lors de l'initialisation de Vuforia
if (e != null) {
    Log.d("VUFORIA_LEGO", "onInitARDone "+e.getString());
    return;
}

// Create OpenGL ES view:
int depthSize = 16;
int stencilSize = 0;
boolean translucent = Vuforia.requiresAlpha();

mGlView = new SampleApplicationGLView(this);
mGlView.init(translucent, depthSize, stencilSize);
```

```
// création de notre renderer et association OpenGL - Renderer
mRenderer = new ARRender(vuforiaAppSession, mTextures);
mGlView.setRenderer(mRenderer);


// ajout de la vue dans la scène
// contentView est le RelativeLayout de l'application
RelativeLayout contentView = (RelativeLayout)
findViewById(R.id.activity_lego_r);
// on enlève la progress bar
contentView.removeView(findViewById(R.id.loading_indicator));
// ajout de la scène openGL, par ligne de code java, et ainsi pour éviter
des marges...
addContentView(mGlView, new
ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,ViewGroup.Layout
Params.MATCH_PARENT));
// on rend la vue initiale transparente. Elle reste car on peut gérer des
touch avec
contentView.setBackgroundColor(Color.TRANSPARENT);
contentView.bringToFront();
```

Pour l'instant, cela ne fera rien, juste une vue (le renderer) « noire ». La reconnaissance n'est pas encore lancée.

3e étape : lancer l'environnement Vuforia

Pour lancer cette reconnaissance, il faudra aussi prévoir de l'arrêter et de relancer la reconnaissance et la scène openGL via les méthodes onPause et onResume. Pour terminer l'application, quand le système Android détruira l'application, il faut aussi implémenter onDestroy :

```
// Called when the activity will start interacting with the user.
@Override
protected void onResume()
{
    super.onResume();


    try
    {
        vuforiaAppSession.resumeAR();
    } catch (SampleApplicationException e) { }

    // Resume the GL view:
    if (mGlView != null)
    {
        mGlView.setVisibility(View.VISIBLE);
        mGlView.onResume();
    }

}


// Called when the system is about to start resuming a previous activity.
@Override
protected void onPause()
{
    super.onPause();

    if (mGlView != null)
    {
```

```
        mGlView.setVisibility(View.INVISIBLE);
        mGlView.onPause();
    }

    try
    {
        vuforiaAppSession.pauseAR();
    } catch (SampleApplicationException e)  {  }
}

// The final call you receive before your activity is destroyed.
@Override
protected void onDestroy()
{
    super.onDestroy();

    try
    {
        vuforiaAppSession.stopAR();
    } catch (SampleApplicationException e) { }

    // Unload texture:
    mTextures.clear();
    mTextures = null;

    System.gc();
}
```

Pour lancer l'environnement Vuforia, dans lequel seront faites les reconnaissances, il faut ajouter à la fin de onInitARDone le code suivant (pour l'instant rien ne sera reconnu puisque les méthodes doInitTrackers et doLoadTackersData sont encore vides) :

```
try
{
vuforiaAppSession.startAR(CameraDevice.CAMERA_DIRECTION.CAMERA_DIRECTION_DE
FAULT);
} catch (SampleApplicationException ex)
{
    Log.e("VUFORIA_LEGO", ex.getString());
}
```

Cette fois-ci vous devez voir le flux caméra dans votre application.

4e étape : préparation de la reconnaissance de marker

Pour la reconnaissance via Vuforia, vous utlisez des « FrameMarker » (des marqueurs encadrés, des sortes de QR Code). Ces FrameMarkers sont intégrés à Vuforia, ce qui simplifie l'utilisation. Il faut commencer par initialiser le « Trackes », ce qui reconnaitra les Markers. Ici tout ce passe en interne à Vuforia :

```
@Override
public boolean doInitTrackers() {
    // Indicate if the trackers were initialized correctly
    boolean result = true;

    // Initialize the marker tracker:
    TrackerManager trackerManager = TrackerManager.getInstance();
```

```
    Tracker trackerBase =
trackerManager.initTracker(MarkerTracker.getClassType());
    MarkerTracker markerTracker = (MarkerTracker) (trackerBase);

    if (markerTracker == null)
    {
        Log.e( "VUFORIA_LEGO","Tracker not initialized. Tracker already
initialized or the camera is already started");
        result = false;
    } else
    {
        Log.i("VUFORIA_LEGO", "Tracker successfully initialized");
    }

    return result;
}
```

Une fois le Tracker initialisé, il faut encore créer les Markers correspondants. Pour la méthode createFrameMarker, la chaine de caractère est juste un nom qui vous permettra de savoir quel marker est reconnu :

```
@Override
public boolean doLoadTrackersData() {
    Log.d("VUFORIA_LEGO", "doLoadTrackersData");
    TrackerManager tManager = TrackerManager.getInstance();
    MarkerTracker markerTracker = (MarkerTracker) tManager
            .getTracker(MarkerTracker.getClassType());
    if (markerTracker == null)
        return false;

    dataSet = new Marker[4];

    dataSet[0] = markerTracker.createFrameMarker(0, "ETAPE01", new
Vec2F(50, 50));
    if (dataSet[0] == null)
    {
        Log.i("VUFORIA_LEGO", "ETAPE01 Failed.");
        return false;
    }

    dataSet[1] = markerTracker.createFrameMarker(1, "ETAPE02", new
Vec2F(50, 50));
    if (dataSet[1] == null)
    {
        return false;
    }

    dataSet[2] = markerTracker.createFrameMarker(2, "ETAPE03", new
Vec2F(50, 50));
    if (dataSet[2] == null)
    {
        return false;
    }

    dataSet[3] = markerTracker.createFrameMarker(3, "ETAPE04", new
Vec2F(50, 50));
    if (dataSet[3] == null)
    {
        return false;
    }
```

```
    Log.i("VUFORIA_LEGO", "Successfully initialized MarkerTracker.");

    return true;
}
```

5e étape : lancer la reconnaissance

A ce stade, le Tracker n'est pas encore lancé. Pour cela il faut implémenter la méthode doStartTrackers et la méthode associée doStopTrackers :

```
@Override
public boolean doStartTrackers() {
    Log.d("VUFORIA_LEGO", "doStartTrackers");
    // Indicate if the trackers were started correctly
    boolean result = true;

    TrackerManager tManager = TrackerManager.getInstance();
    MarkerTracker markerTracker = (MarkerTracker)
tManager.getTracker(MarkerTracker.getClassType());
    if (markerTracker != null)
        markerTracker.start();

    return result;
}

@Override
public boolean doStopTrackers() {
    Log.d("VUFORIA_LEGO", "doStopTrackers");
    boolean result = true;

    TrackerManager tManager = TrackerManager.getInstance();
    MarkerTracker markerTracker = (MarkerTracker)
tManager.getTracker(MarkerTracker.getClassType());
    if (markerTracker != null)
        markerTracker.stop();

    return result;
}
```

Un fois encore, même si la reconnaissance est faite par Vuforia, il ne se passe rien dans l'application. Avant de modifier le Renderer, il reste deux méthodes à implémenter : doUnloadTrackersData et doDeinitTrackers. Ces méthodes servent pour quitter « proprement ». Ici, comme les Markers sont des objets internes à Vuforia, il n'y a rien à faire dans doUnloadTrackersData, si ce n'est retourné « true ». Dans doDeinitTrackers, il suffit ici d'appeler des méthodes de Vuforia pour désactiver le Tracker de FrameMarker :

```
@Override
public boolean doUnloadTrackersData() {
    Log.d("VUFORIA_LEGO", "doUnloadTrackersData");
    return true;
}

@Override
public boolean doDeinitTrackers() {
    Log.d("VUFORIA_LEGO", "doDeinitTrackers");
    TrackerManager tManager = TrackerManager.getInstance();
    tManager.deinitTracker(MarkerTracker.getClassType());
    return false;
}
```

6e étape : traiter la reconnaissance (1/2)

Pour la prise en compte des markers, cela se fera dans le Renderer dans la méthode onDrawFrame. A travers un objet « State », il est possible d'avoir les Marker reconnus. L'objet state de la version fournie de ARRenderer est l'objet qu'il faut exploiter. Dans onDrawFrame ajoutez le code suivant (juste avant la dernière ligne « Renderer.getInstance().end(); ») :

```
for (int tIdx = 0; tIdx < state.getNumTrackableResults(); tIdx++) {
            // Get the trackable:
            TrackableResult trackableResult =
state.getTrackableResult(tIdx);


            // Check the type of the trackable:
            // ici on s'assure de ne traiter que des FrameMarker
            assert (trackableResult.getType() ==
MarkerTracker.getClassType());
            MarkerResult markerResult = (MarkerResult) (trackableResult);
            Marker marker = (Marker) markerResult.getTrackable();

            Log.d("VUFORIA_LEGO", "marker id = "+marker.getMarkerId()+" et
nom = "+marker.getName());
        }
```

Vous avez maintenant une trace dans le log cat de la reconnaissance des markers.

7e étape : dessiner en fonction de la reconnaissance (2/2)

Pour le dessin en lui-même, il faut utiliser les capacités de Vuforia avec les matrices de transformations (entre les repères caméra / marker / écran).

La barre de titre de l'application entraine aussi un décale, pour éviter de devoir calculer une translation (qui dépend de la hauteur de cette barre), le plus simple est dans l'enlever. Dans res/values/styles.xml changer le thème pour un thème sans barre, par exemple :

```
<style name="AppTheme" parent="android:Theme.Holo.Light.NoActionBar">
```

Pour dessiner une image en 2D, le principe est de plaquer une texture sur une surface. Il faut donc une zone où dessiner (la surface) données par un jeu de coordonnées (x,y,z) , la texture (l'image) et les coodonnées dans l'image pour dire ce qui est affiché. Vous pouvez vous inspirer de (attention, c'est du c++) : https://library.vuforia.com/articles/Solution/How-To-Draw-a-2D-image-on-top-of-a-target-using-OpenGL-ES

Pour l'adaptation, ici cela donne (pour la méthode onDrawFrame) en deux temps : des déclarations en plus (pour donner les coordonnées) et la fonction onDrawFrame :

```
float planeVertices[] =
    {
            -0.5f, -0.5f, 0.0f, 0.5f, -0.5f, 0.0f, 0.5f, 0.5f, 0.0f, -0.5f,
0.5f, 0.0f,
    };
float planeTexcoords[] =
    {
            0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f
    };
```

```
float planeNormals[] =
    {
            0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
0.0f, 1.0f
    };
 short planeIndices[] =
    {
            0, 1, 2, 0, 2, 3
    };


Buffer bufferPlaneVertices = fillBuffer(planeVertices);
Buffer bufferPlaneNormals = fillBuffer(planeNormals);
Buffer bufferPlaneTexcoords = fillBuffer(planeTexcoords);
Buffer bufferPlaneIndices = fillBuffer(planeIndices);


@Override
public void onDrawFrame(GL10 gl) {
    // init de la scène avec la video...
    // Clear color and depth buffer
    GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT |
GLES20.GL_DEPTH_BUFFER_BIT);

    // Get the state from Vuforia and mark the beginning of a rendering
    // section
    State state = Renderer.getInstance().begin();

    // Explicitly render the Video Background
    Renderer.getInstance().drawVideoBackground();

    GLES20.glEnable(GLES20.GL_DEPTH_TEST);
    // We must detect if background reflection is active and adjust the
    // culling direction.
    // If the reflection is active, this means the post matrix has been
    // reflected as well,
    // therefore standard counter clockwise face culling will result in
    // "inside out" models.
    GLES20.glEnable(GLES20.GL_CULL_FACE);
    GLES20.glCullFace(GLES20.GL_BACK);
    if (Renderer.getInstance().getVideoBackgroundConfig().getReflection()
== VIDEO_BACKGROUND_REFLECTION.VIDEO_BACKGROUND_REFLECTION_ON)
        GLES20.glFrontFace(GLES20.GL_CW);  // Front camera
    else
        GLES20.glFrontFace(GLES20.GL_CCW);   // Back camera

    // Set the viewport
    int[] viewport = vuforiaAppSession.getViewport();
    GLES20.glViewport(viewport[0], viewport[1], viewport[2], viewport[3]);

    for (int tIdx = 0; tIdx < state.getNumTrackableResults(); tIdx++) {
        // Get the trackable:
        TrackableResult trackableResult = state.getTrackableResult(tIdx);


        // Check the type of the trackable:
        // ici on s'assure de ne traiter que des FrameMarker
        assert (trackableResult.getType() == MarkerTracker.getClassType());
        MarkerResult markerResult = (MarkerResult) (trackableResult);
        Marker marker = (Marker) markerResult.getTrackable();
```

```
        Log.d("VUFORIA_LEGO", "marker id = "+marker.getMarkerId()+" et nom
= "+marker.getName());

        // Choose the texture based on the target id :
        int textureIndex = 0;
        textureIndex = marker.getMarkerId();
        assert (textureIndex < mTextures.size());
        Texture thisTexture = mTextures.get(textureIndex);

        // calcul géométrique :
        float[] modelViewMatrix = Tool.convertPose2GLMatrix(
                trackableResult.getPose()).getData();
        float[] modelViewProjection = new float[16];
        Matrix.translateM(modelViewMatrix, 0, 0f, 0f, 0f);
        Matrix.scaleM(modelViewMatrix, 0, marker.getSize().getData()[0],
marker.getSize().getData()[1], 1f);
        Matrix.multiplyMM(modelViewProjection, 0, vuforiaAppSession
                .getProjectionMatrix().getData(), 0, modelViewMatrix, 0);

        // préparation openGL
        GLES20.glUseProgram(shaderProgramID);
        GLES20.glEnableVertexAttribArray(vertexHandle);
        GLES20.glEnableVertexAttribArray(textureCoordHandle);
        GLES20.glVertexAttribPointer(vertexHandle, 3, GLES20.GL_FLOAT,
false, 0, bufferPlaneVertices);
        GLES20.glVertexAttribPointer(normalHandle, 3, GLES20.GL_FLOAT,
false, 0, bufferPlaneNormals);
        GLES20.glVertexAttribPointer(textureCoordHandle, 2,
GLES20.GL_FLOAT, false, 0, bufferPlaneTexcoords);

        GLES20.glEnableVertexAttribArray(vertexHandle);
        GLES20.glEnableVertexAttribArray(normalHandle);
        GLES20.glEnableVertexAttribArray(textureCoordHandle);

        GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D,
thisTexture.mTextureID[0]);
        GLES20.glUniformMatrix4fv(mvpMatrixHandle, 1, false,
modelViewProjection,0 );
        GLES20.glDrawElements(GLES20.GL_TRIANGLES, 6,
GLES20.GL_UNSIGNED_SHORT, bufferPlaneIndices);


        // ce qui a été activé doit être désactivé
        GLES20.glDisableVertexAttribArray(vertexHandle);
        GLES20.glDisableVertexAttribArray(normalHandle);
        GLES20.glDisableVertexAttribArray(textureCoordHandle);

    }

    GLES20.glDisable(GLES20.GL_DEPTH_TEST);
    Renderer.getInstance().end();
}
```

8e étape : pour aller plus loin

L'aspect programmation ne suffit pas : les marqueurs peuvent contenir une image, si elle est bien choisie et que le marqueur est bien positionné, vous avez alors une application de réalité augmentée.

Maintenant vous pouvez adapter ce tutoriel. Certaines pièces apparaissent dans deux étapes : afficher les deux pour les pièces en question, soit en même temps, soit avec une « gesture » pour passer d'une étape à une autre.