Fisheye View – 2ième partie – Multimodalité – version Java/Android

Dans ce TD, nous voyons la partie implémentation, mais vous voyez qu'en tant que concepteur, vous avez alors beaucoup de choix dans les interactions…

La multi-modalité permet donc de combiner des modalités pour obtenir une interaction enrichie. Cet enrichissement peut être :

- De la souplesse d'interaction (choix entre plusieurs modalités qui sont alors concurrente)
- Du contrôle dans l'interaction. En effet, certaines modalités (accéléromètres, reconnaissances vocales, etc.) peuvent ne pas être « pratiques » si elles sont activées en permanence… alors on peut les combiner à une autre modalité (déclencheur) pour les utiliser sur demande. Par exemple, dans le cas de l'interaction 2, on peut ajouter un bouton qui déclenche une « reconnaissance » (les deux modalités sont alors séquentielle et complémentaire), ou alors on peut aussi ajouter un bouton qu'il faut maintenir « enfoncé » pour activer la reconnaissance (les deux modalités sont alors redondante).
- De la complexité d'interaction, en combinant plusieurs actions en une. Par exemple, en combinant les interactions 1 et 2, on pourrait avoir la reconnaissance d'une action (« zoomer » , « déplacer », « agrandir », etc.) et le mouvement (gauche / droite) donnerait l'amplitude de l'action (par exemple un mouvement à gauche serait un petit zoom, un mouvement à droite un grand zoom) ou le sens de l'action (par exemple un mouvement à gauche serait un zoom « in », un mouvement à droite un zoom « out »).

La figure 1, extraite de la thèse de Frédéric Vernier http://iihm.imag.fr/publs/2001/These_Vernier.pdf , montre les différentes combinaisons possibles selon différents critères. Une fois encore, cela montre la complexité de la conception… et donc de l'évaluation et de la programmation…

Schémas de composition

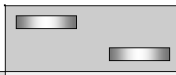
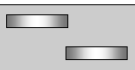
Aspects de composition \ Composition					
Temporelle	Anachronique	Séquentielle	Concomitante	Coïncidente	Parallèle / Simultanée
Spatiale	Disjointe	Adjacente	Intersectée	Imbriquée	Recouvrance
Articulatoire	Indépendance	Fissionnée	Fissionnée + Dupliquée	Partiellement Dupliquée	Dupliquée
Syntaxique	Différente	Complétion	Divergence	Extension	Jumelage
Sémantique	Concurrente	Complé-mentaire	Complémentaire + Redondante	Partiellement Redondante	Totalement Redondante

Tableau 1 : Application des schémas de composition aux cinq aspects proposés.

Figure 1. extrait de la thèse de Frédéric Vernier

Travail demandé : implémenter quelques (2, 3, …) modalités pour contrôler la Fisheye View (centre, paramètres z-r-o, activation / désactiviation, etc.)

Ce qui vous est demandé de faire, c'est expérimenter en termes de programmation mais aussi de réglage, des techniques d'interactions différentes (au moins équivalente). Vous pourrez par exemple faire essayer vos techniques à vos camarades.

A vous d'inventer vos modalités, en relation avec vos expérimentations à venir !

Ci-dessous vous trouverez des indications pour 6 moyens d'interaction (dispositifs) à vous d'imaginer les langages (ou à choisir les tâches) que vous voulez effectuer avec.

Une fois quelques interactions implémentées, essayer d'en combiner.

Touch

Android fournit une interface de base, mais en interprétant les informations, il est possible de faire différentes interactions. Par exemple, décaler le point de « touch » pour manipuler quelque chose qui n'est pas sur le doigt. Autre exemple, fournir des zones verticales et horizontales en dehors de l'image pour déplacer le fisheye.

C'est un cas assez simple, car il faut avoir un « listener » de type « View.OnTouchListener » , d'implémenter la méthode et d'utiliser la méthode setOnClickListener sur la « view » dont vous voulez récupérer les événements de type « touch » :

public abstract boolean onTouch (View v, MotionEvent event)

Called when a touch event is dispatched to a view. This allows listeners to get a chance to respond before the target view.

Parameters

- **v** The view the touch event has been dispatched to.

event The MotionEvent object containing full information about the event.

Returns

- True if the listener has consumed the event, false otherwise.

Il faut retourner "true" si vous voulez pouvoir suivre le déplacement (équivalent du drag).

MultiTouch

C'est une exploitation complémentaire du Touch.

Le MotionEvent (e) propose des méthodes pour savoir combien il y a de contacts reconnus (*Pointer*). A partir de là, les informations pour chaque *pointer* sont exploitables. Les méthodes à retenir sont :

- e.getActionMasked() qui permet de savoir quel type d'événement a été détecté (appui, déplacement, "relevage" [*up*]).
- e.getActionIndex() pour savoir quel *pointer* est concerné par l'événement. Chaque pointer a un id (un entier), dans l'ordre de détection, de 0 à ..., du plus ancien... avec une réindexation (si le 2de *pointer* est enlevé (*up*), il avait l'indice 1. Le troisième devient de 2de et il change d'index, il passe de 2 à 1)
- e.getPointerId(*index*) permet d'avoir l'id du *pointer* qui lui ne changera pas jusqu'à sa disparition
- toutes les méthodes de MotionEvent qui prennent un index (ou un id) en paramètre

Accelerometer

Il s'agit de capter les accélérations selon 3 axes. Ces 3 axes sont définis en fonction du dispositif (et leur interprétation dépend donc de l'orientation du dispositif). Ce sont donc des accélérations, qui incluent l'accélération constante qu'est la gravité. Ce sont des $m.s^{-2}$, dont il faut déterminer ce qu'il convient d'en faire. Il est possible de chercher à respecter les lois de la physique, mais il est plus simple de faire sa propre interprétation.

Exemple d'utilisation :

Etape 1 : abonnement et variabls nécessaire (ici dans l'activité)

```
WindowManager wm = (WindowManager) getSystemService(Context.WINDOW_SERVICE);
Display display = wm.getDefaultDisplay();
SensorManager mSM = (SensorManager)      getSystemService(Context.SENSOR_SERVICE);
Sensor mAccelerometer = mSM.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Etape 2 : lancer observation des capteurs

```
      mSM.registerListener(aSensorEventListener, mAccelerometer,
SensorManager.SENSOR_DELAY_UI);
```

Etape 3 : récupérer les éléments (deux méthodes, onAccuracyChanged et onSensorChanged), avec prise en compte de la rotation...

```
public void onSensorChanged(SensorEvent sensorEvent) {
        if (sensorEvent.sensor.getType() != Sensor.TYPE_ACCELEROMETER)
            return;

        switch (display.getRotation()) {
            case Surface.ROTATION_0:
                x = sensorEvent.values[0];
                y = sensorEvent.values[1];
                break;
            case Surface.ROTATION_90:
                x = -sensorEvent.values[1];
                y = sensorEvent.values[0];
                break;
            case Surface.ROTATION_180:
                x = -sensorEvent.values[0];
                y = -sensorEvent.values[1];
                break;
            case Surface.ROTATION_270:
                x = sensorEvent.values[1];
                y = -sensorEvent.values[0];
                break;
        }
        // z = sensorEvent.values[2];

        // à vous de gérer x, y, z... // et de faire le langage de la modalité
    }
```

Etape 4: il ne faut pas oublier de se désabonner lorsque l'activité n'est plus au-dessus de la pile des activités (n'est plus affichée)

```
mSM.unregisterListener(aSensorEventListener);
```

Accelerometre et Champs magnétique

Un exemple plus complet d'utilisation de ces capteurs est disponible dans le dossier android-sdk\samples\android-10\AccelerometerPlay

Le principe est le suivant : on s'abonne à un type d'événement à un SensorManager, il faut ensuite traiter les événements reçus.

Voici le code dont vous aurez besoin :

Etape 1 : declaration

```
	// Il faudra implémenter l'interface :
	// SensorEventListener

	// variable nécessaire pour s'abonner
	private SensorManager mSensorManager;
	private Sensor mAccelerometer;
```

Etape 2 : retrouver le capteur fourni par le système

```
// ce code est extrait d'une Activity
// Là c'est pour savoir les changements d'orientation du téléphone
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
// pour obtenir la « gravité »
mAccelerometer=mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
// pour obtenir le champ magnétique
mMagnetic = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

Etape 3 : s'abonner par exemple quand l'application est réactivée

```
	protected void onResume() {
		super.onResume();
		// on s'abonne en précisant le listener, le capteur
		// et le type de fréquence de réception des événements
		mSensorManager.registerListener(this, mAccelerometer,
SensorManager.SENSOR_DELAY_UI);
		mSensorManager.registerListener(this, mMagnetic,
SensorManager.SENSOR_DELAY_FASTEST);

	}
```

Etape 3 bis : on doit se désabonner quand on quitte l'application

```
	// ce code est extrait d'une Activity
	protected void onPause() {
		super.onPause();
		mSensorManager.unregisterListener(this);
	}

	@Override
	protected void onStop() {
		super.onStop();
		mSensorManager.unregisterListener(this);

	}
```

Etape 4 : on implémente `SensorEventListener`

```
public void onAccuracyChanged(Sensor sensor, int accuracy) {
	}


float [] gravity = new float[3];
float [] geomagetic = new float[3];

public void onSensorChanged(SensorEvent event) {
		// pour obtenir l'orientation, il faut avoir des
		// matrices de rotation et d'inclinaison
		// pour les avoir, il faut la gravité et le champ magnétique

		if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
				for(int i=0; i<3; i++){
				geomagetic[i] =  event.values[i];
			}
		}

		if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
				for(int i=0; i<3; i++){
				gravity[i] =  event.values[i];
			}
		}


		if ((gravity[2] == 0) || (geomagetic[0] == 0))
			{
			// on n'a pas les infos pour calculer les matrices
			return;
			}

		// calcul des matrices
		float[] RotationMatrix = new float[9];
		float[] I = new float[9];
		SensorManager.getRotationMatrix(RotationMatrix, I, gravity,
geomagetic);

		// calcul de l'orientatoin
		float [] values = new float[3];
		SensorManager.getOrientation(RotationMatrix, values);

		// il reste à faire le traitement de l'orientation
	}
```

Le détail du contenu d'un sensorevent est décrit à :

http://developer.android.com/reference/android/hardware/SensorEvent.html

VoiceRecognition

Pour la reconnaissance vocale, il est facile d'appeler un web service (fourni par android) qui va analyser le son. Il faudra donc le réseau d'activer.

En reprenant du code de l'exemple VoiceRecognition (dans le dossier <android-sdk>/samples/android-8/VoiceRecognitionService), qui déclenche la reconnaissance suite à un « clic » sur un bouton, voici comme vous y prendre :

Etape 1 : vérification de la disponibilité

```
        // Get display items for later interaction
        // btn_speak est un bouton définit dans le fichier xml du layout
          Button speakButton = (Button) findViewById(R.id.btn_speak);

          // Check to see if a recognition activity is present
          PackageManager pm = getPackageManager();
          List<ResolveInfo> activities = pm.queryIntentActivities(
                  new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);
          // activities.size() >  0 => la reconnaissance est disponible
          if (activities.size() != 0) {
              speakButton.setOnClickListener(this);
          } else {
          // pas de reconnaissance, on désactive le déclencheur
              speakButton.setEnabled(false);
              speakButton.setText("Recognizer not present");
          }
```

Etape 2 : écouter les événements sur le bouton déclencheur

public void onClick(View v) {

```
        // si on a cliqué sur le bouton
        if (v.getId() == R.id.btn_speak) {
         // création d'une nouvelle opération, opération de reconnaissance
         Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
         // paramétrage pour une reconnaissance du langage naturel
         intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                 RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
         // un texte particulier pour la boite de dialogue
          intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Fisheye - Speech
recognition demo");
         // on lance la reconnaissance
         // VOICE_RECOGNITION_REQUEST_CODE est un code à nous pour
         // identifier la réponse
          startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);        }

}
```

Etape 3 : il faut avoir défini une methode onActivityResult pour recevoir la réponse

```
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
         // on vérifie que la réponse est bonne et pour nous
         // RESULT_OK est une constante de la classe Activity
         // ce code extrait était placé dans une Activity
          if (requestCode == VOICE_RECOGNITION_REQUEST_CODE && resultCode ==
RESULT_OK) {
             // on récupère les réponses
```

```
        ArrayList<String> matches = data.getStringArrayListExtra(
                RecognizerIntent.EXTRA_RESULTS);

        int nb_rep = matches.size();
        if (nb_rep > 0)
        {
              // par exemple ici, s'il y en a, on s'intéresse aux
              // 3 premières réponses pour ne pas être trop éloigné
              int nb_test = Math.min(3, nb_rep);

              for(int i = 0; i < nb_test; i++)
              {
                    if (matches.get(0).toLowerCase().equals("zoomer"))
                    {
                          // code à exécuter quand on a dit « zoomer »
                          break;
                    }
                    // etc.                 }
        }
    }

  // ce code extrait était placé dans une Activity
  super.onActivityResult(requestCode, resultCode, data);
}
```

Gesture

Il y a deux reconnaissances de gestes proposées par android :

- GestureDetectorCompat pour des gestes comme le *scroll (swipe)*, les appuis longs, les "double tocuh", etc.
- ScaleGestureDetector pour le *pinch close/open* (resserer ou écarter deux doigts pour zoomer par exemple)

Même si ces classes n'ont pas de lien (pas d'héritage commun autre que Object), elles fonctionnent de la même manière : il faut les appeler avec un MotionEvent et elle rappelle en callback via des interfaces qu'elles proposent (ScaleGestureDetector.OnScaleGestureListener, GestureDetector.OnGestureListener, GestureDetector.OnDoubleTapListener).

Exemple, écoutes de geste sur une view

Etape 1 : création des objets

```
View v = findViewById(R.id.control);

gestures = new GestureCommad(this, [...]); // création pour reconnaissance de geste
scales = new ScaleCommand(this, [...]);// création pour reconnaissance de scale
// this = l'activité = le context

// fusion des deux OnClickListener
v.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View view, MotionEvent motionEvent) {
             // appel des onTouch, à l'intérieur desquels chaque Detector
             // spécifique sera appelé
                gestures.onTouch(view, motionEvent);
                scales.onTouch(view, motionEvent);
                return true;
            }
        });
```

Etape 2 : initialisation des objets de la reconnaissance de geste

```
public class ScaleCommand implements ScaleGestureDetector.OnScaleGestureListener,
View.OnTouchListener {

    ScaleGestureDetector scaleDetector;
    public ScaleCommand(Context c, [...]) {
        scaleDetector = new ScaleGestureDetector(c,this);
       // les autres paramètres servent pour le langage
       // l'interface OnTouchListener n'est pas forcément utiliée. Elle peut l'être
       // si c'est le seul détecteur... mais s'il y en a deux, il n'en faut qu'une
       // d'où la fusion ci-dessus
    }

    @Override
    public boolean onScale(ScaleGestureDetector scaleGestureDetector) {
       // langage à appliquer...
       // scaleGestureDetector.getScaleFactor() fournit l'échelle
        return true;
    }

    @Override
```

```
    public boolean onScaleBegin(ScaleGestureDetector scaleGestureDetector) {
       // même si rien n'est fait ici, il faut retourner true pour que la suite,
       // i.e., le onScale, puisse se produire
        return true;
    }

    @Override
    public void onScaleEnd(ScaleGestureDetector scaleGestureDetector) {
    }

    @Override
    public boolean onTouch(View view, MotionEvent motionEvent) {
        // Appel au détecteur qui rappellera en callback...
        scaleDetector.onTouchEvent(motionEvent);
        return true;
    }
}
```