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

Lors du tp précédent, nous avons mis en place un fisheye avec les formules de Volkmar Hovestadt :

$$FDeform(x) = x \times \frac{\sqrt{(x^2+z^2)(r^2-(z-o)^2)+z^2(z-o)^2}+z(z-o)}{x^2+z^2}$$

$$FDeform^{-1} = \frac{z \times x}{\sqrt{r^2-x^2}+(z-o)}$$

La fonction FDeform permet de transformer l'espace : FDeform(0) = 0 (pas de déformation du centre) pour x proche de 0, FDeform(x) sera plus grande (en valeur absolue) et FDeform tend vers une valeur finie quand x tend vers l'infini.

x est une distance en Pixel avec le centre de la déformation.

Il n'y a pas qu'une seule formule de Fisheye View. Implémentez aussi d'autres formules, par exemple une formule simplifiée (c.f. thèse de Frédéric Vernier - http://iihm.imag.fr/publs/2001/These_Vernier.pdf)

$$FDeform(x) = \frac{R \times Z}{-Z-x} + R$$

$$FDeform^{-1}(x) = \frac{R \times Z}{R-x} - Z$$

Le but de cette séance est de voir comment exploiter les capacités d'interaction des dispositifs android. L'objectif pédagogique n'est pas d'apprendre à programmer, ce que vous savez déjà faire, mais de réfléchir en termes de techniques d'interactions.

Avant toutes autres considérations, nous avons besoins d'introduire la multimodalité.

Introduction à la multimodalité

Définition d'une modalité

En suivant la définition proposée par Laurence Nigay, nous verrons une modalité comme :

<dispositif, langage>

Le dispositif est la source de l'information, le langage son interprétation pour la traduire en action utilisateur sur le système, par exemple :

1. Considérons une application sur un téléphone qui propose une fisheye view. Lorsque je touche l'écran et que je vais sur la gauche, cela agrandit la zone de la déformation. Lorsque

je touche l'écran et que je vais sur la droite, cela réduit la zone de la déformation.
Dans un tel cas, le dispositif est l'écran tactile (et la génération d'événement), le langage est l'interprétation des valeurs reçues pour avoir un geste vers la gauche ou vers la droite.

2. Toujours sur cette même application, je peux utiliser des commandes vocales pour faire quelques actions : dire « zoomer » pour augmenter le grossissement ou dire « reculer » pour diminuer le grossissement. Dans un tel cas, le dispositif est le micro, le langage est un « langage pseudo-naturel » (un sous ensemble limité du langage)
3. Une variante de cette application : lorsque je touche l'écran cela déplace le centre de la zone de déformation. Le dispositif est l'écran tactile (et la génération d'événement), le langage est le changement du centre de la déformation…

Combinatoire pour une modalité : difficulté de conception

Ainsi, il y a de grandes possibilités d'interaction :

- Pour un même couple <dispositif, langage>, il peut y avoir des réglages et des paramétrages pour chaque éléments. Dans le cas de l'interaction 1, on pourrait par exemple régler la sensibilité de l'écran tactile, la fréquence d'émission d'événement. On pourrait aussi régler la sensibilité du déplacement vers la gauche : 1 pixel de différence ? 10 ? plus ? pour détecter un mouvement vers la gauche ou vers la droite ?
  Ainsi pour une modalité, la combinatoire des paramètres peut la changer grandement et la rendre utilisable ou inutilisable.
- Derrière un langage d'une modalité, il y a à priori une tâche. L'appariement tâche-dispositif est du domaine de la conception (c.f. CEIHM). C'est-à-dire qu'en changeant le langage d'interaction associé à un dispositif, on en change l'utilisation et l'activité réalisable avec le dispositif. Si l'on compare les interactions 1 et 3, seul le langage change…
  Ainsi, dans le cadre d'une application, il existe une combinatoire importante entre les dispositifs utilisés et les langages, ce qui amène à des solutions différentes de conception.
- Pour une interaction donnée, le contexte peut aussi amener à changer les paramètres. Par exemple pour l'interaction 2, si je suis en train de parler à quelqu'un, le système pourrait capter des mots et les interpréter. Il faudrait alors pouvoir désactiver des modalités dans certaines situations, et donc, si on veut pouvoir continuer à utiliser les fonctionnalités, il faudrait associer une autre modalité à la fonctionnalité.
  Ainsi, il existe aussi une combinatoire entre les dispositifs, les langages et les situations d'usages.

Le dernier point conduit naturellement à la multi-modalité, dont l'essence est de combiner des modalités pour obtenir de nouvelles interactions (ce qui augmentent encore la combinatoire…).

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…

Définition de la multi-modalité

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 »).

Le tableau ci-dessous, extrait de la thèse de Frédéric Vernier - http://iihm.imag.fr/publs/2001/These_Vernier.pdf , monter 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

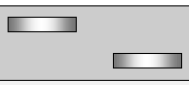
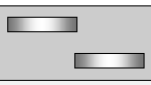
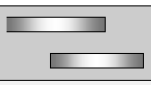
Aspects de composition \ Composition					
Temporelle	Anachronique	Séquentielle	Concomitante	Coïncidente	Parallèle / Simultanée
Spatiale	Disjointe	Adjacente	Intersectée	Imbriquée	Recouvrance
Articulatoire	Indépendance	Fissionnée	Fissionnée + Dupliquée	Partiellement Dupliquée	Dupliquée
Syntaxique	Différente	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 terme de programmation mais aussi de réglage, des techniques d'interactions différentes, voire d'essayer de les combiner. Vous pourrez par exemple faire essayer vos techniques à vos camarades.

A vous d'inventer vos modalités !

Ci-dessous vous trouverez des indications pour 4 moyens d'interaction (dispositifs) à vous d'imaginer les langages (ou à choisir les tâches) que vous voulez effectuer avec.

Touch

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).

TrackBall

Lorsque l'on définit une « View » personnalisée, il est possible d'implémenter la méthode :

public boolean onTrackballEvent (MotionEvent event)

Implement this method to handle trackball motion events. The *relative* movement of the trackball since the last event can be retrieve with `MotionEvent.getX()` and `MotionEvent.getY()`. […]

Parameters

event The motion event.

Returns

- True if the event was handled, false otherwise

Pour que l'on puisse récupérer les événements, il faut que la "view" ait le focus. Pour cela, on peut spécifier dans le XML : android:focusableInTouchMode="true"
ou dans le code utiliser la méthode setFocusableInTouchMode(true).

De plus méthode requestFocus() de « View » permet de récuper le focus…

Accelerometre

Un exemple plus complet d'utilisation de ces capteurs est disponible dans :
http://developer.android.com/resources/samples/AccelerometerPlay/src/com/example/android/accelerometerplay/AccelerometerPlayActivity.html

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);
mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION);
```

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);

	}
```

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) {
	}


public void onSensorChanged(SensorEvent event) {
		// on peut verifier qu'on a le bon type d'événement
		// c'est important ce que contient event en dépend
		if (event.sensor.getType() != Sensor.TYPE_ORIENTATION)
			return;

		// traitement des données reçues…

	}
```

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 (http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/app/VoiceRecognition.html) , 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);
    }
```