Développement Android (4.3)

Capteurs

BENJAMIN BILLET

WARNING

Le contenu de cette présentation est basé sur la documentation anglophone officielle d'Android, diffusée sous licence *Creative Commons Attribution 2.5* :

developer.android.com

La plupart des schémas qui composent ce cours proviennent de cette documentation et sont, par conséquent, soumis à cette même licence.

http://creativecommons.org/licenses/by/2.5/

Au Sommaire !

- Capteurs environnementaux
- Capteurs de position
- Capteurs de mouvement
- Capteurs multimédias

CAPTEURS ENVIRONNEMENTAUX

Capteurs environnementaux

- Android supporte plusieurs types de capteurs environnementaux.
- TYPE_AMBIENT_TEMPERATURE : température (°C).
- TYPE_LIGHT : illumination (lx).
- TYPE_PRESSURE : pression atmosphérique (mbar ou hPa).
- TYPE_RELATIVE_HUMIDITY : humidité relative (%), i.e., la quantité de vapeur d'eau contenue dans l'air.

CAPTEUR ENVIRONNEMENTAUX

```
SensorManager sensorMgr = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

// list the available sensors
List<Sensor> deviceSensors = sensorMgr.getSensorList(Sensor.TYPE_ALL);

// get a sensor
Sensor sensor = sensorMgr.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE);

if (sensor != null)
   // there is a thermometer
else
   // no thermometer

float consumption = sensor.getPower(); // mA
float resolution = sensor.getResolution();
String vendor = sensor.getVendor();
float range = sensor.getMaximumRange();
```

- Remarque : résolution : le plus petit changement en entrée qui peut être détecté dans le signal de sortie.

UNE SIMPLE ACQUISITION

```
public class MyActivity extends Activity implements SensorEventListener
{
    private SensorManager sensorMgr;
    private Sensor accelerometer;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        ...
        sensorMgr = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        accelerometer = sensorMgr.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy)
    {
        // SENSOR_STATUS_ACCURACY_LOW, SENSOR_STATUS_ACCURACY_MEDIUM, SENSOR_STATUS_ACCURACY_HIGH, SENSOR_STATUS_UNRELIABLE
    }
    @Override
    public void onSensorChanged(SensorEvent event)
    {
        float temperature = event.values[0];
        ...
    }

    @Override
    protected void onResume()
    {
        super.onResume();
        sensorMgr.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_NORMAL);
    }
    @Override
    protected void onPause()
    {
        super.onPause();
        sensorMgr.unregisterListener(this);
    }
}
```

Délai entre chaque mesure

- Lors de l'enregistrement, il est possible de définir un délai entre chaque enregistrement :
  - SENSOR_DELAY_NORMAL : 200 000 µs.
  - SENSOR_DELAY_GAME : 20 000 µs.
  - SENSOR_DELAY_UI : 60,000 µs.
  - SENSOR_DELAY_FASTEST : 0 µs.
  - Ou une valeur personnalisée.
- Ce délai n'est fourni qu'à titre indicatif. Pour une mesure plus précise du temps écoulé entre deux évènements, il est préférable d'utiliser le timestamp du SensorEvent.

Un exemple plus complet

```
private Float temperature;
private Float humidity;
...
sensorMgr.registerListener(this, temperatureSensor, SensorManager.SENSOR_DELAY_NORMAL);
sensorMgr.registerListener(this, humiditySensor, SensorManager.SENSOR_DELAY_NORMAL);
...
@Override
public void onSensorChanged(SensorEvent event)
{
    if(event.sensor == temperatureSensor)
        temperature = event.values[0];
    else if(event.sensor == humiditySensor)
        humidity = event.values[0];

    if(temperature != null && humidity != null)
    {
        float dewPoint = computeDewPoint(temperature, humidity);
        float frostPoint = computeFrostPoint(temperature, dewPoint);
        float absoluteHumidity = computeAbsoluteHumidity();
        float windchill = computeWindChill(someWindValue);
        float humidex = computeHumidex();
        ...
    }
}
```

QUELQUES FORMULES

Le point de rosée est la température à laquelle l'humidité de l'air se condense pour former des gouttelettes d'eau

T : température [0, 60] °C
H_r : humidité relative [0,01 (1 %), 1 (100 %)]
T_r : point de rosée : [0, 50] °C

$$T_r = \frac{b \cdot \lambda(T, H_r)}{a - \lambda(T, H_r)}$$

$$\lambda(T, H_r) = \frac{a \cdot T}{b + T} \cdot \ln(H_r)$$

a = 17.27
b = 237.7 °C

Le point de grivrage est la température à laquelle l'humidité de l'air se condense pour former des cristaux de glace.

T : température < 0° C
T_r : point de givrage (°C)

$$T_g = T_r + \frac{2671.02}{\frac{2954.61}{T} + 2.193665 \ln(T) - 13.3448} - T$$

Quelques formules

L'humidité absolue est la masse de vapeur d'eau contenue dans un volume d'air donné.

ρ : humidité absolue (g/m^3)
T : température (°C)
H_r : humidité relative [0,01 (1 %), 1 (100 %)]

$$\rho = 216.7 \frac{H_r \cdot A \cdot \exp\left(m \cdot \frac{t}{T_n + T}\right)}{273.15 + \mathrm{T}}$$

m = 17.62
T_n = 243.12 °C
A = 6.112 hPa

L'humidité absolue est la masse de vapeur d'eau contenue dans un volume d'air donné.

T = température (°C)
T_r : point de rosée (°C)

$$humidex = T + 0.5555(\alpha - 10)$$

$$\alpha = 6.11 \exp\left(5417.7530\left(\frac{1}{273.16} - \frac{1}{T_r + 273.15}\right)\right)$$

CAPTEURS DE POSITION

Position

- Android fournit trois modes de positionnement :
  - Le positionnement GPS.
    - Précis
    - Coûteux en énergie.
  - Le positionnement réseau, en utilisant la position des réseaux proches (cellule GSM ou bornes WiFi).
    - Moins précis.
    - Moins coûteux en énergie.
  - Le positionnement passif, qui réutilise les valeurs acquises par les autres applications.
    - Précision variable (dépend des autres applications).
    - Ne coûte rien de plus en énergie.

POSITION

```
public class MyActivity extends Activity
{
    private LocationManager locMgr;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        locMgr = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        locMgr.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 10000, 300, myLocationListener);
        locMgr.requestLocationUpdates(LocationManager.GPS_PROVIDER, 10000, 300, myLocationListener);
    }

    @Override
    protected void onDestroy()
    {
        locMgr.removeUpdates(myLocationListener);
        super.onDestroy();
    }
}
```

- Il est possible de spécifier un temps (ms) ou un déplacement (m) minimum pour qu'une nouvelle valeur de position soit transmise à l'application.
- Ne pas oublier de désactiver le listener, sinon l'application continuera à recevoir des notifications en background.

POSITION

```
LocationListener myLocationListener = new LocationListener()
{
    @Override
    public void onLocationChanged(Location location)
    {
    }

    @Override
    public void onProviderDisabled(String provider)
    {
    }

    @Override
    public void onProviderEnabled(String provider)
    {
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras)
    {
        // status : LocationProvider.AVAILABLE, LocationProvider.OUT_OF_SERVICE, LocationProvider.TEMPORARILY_UNAVAILABLE
    }
};
```

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

- L'accès au GPS nécessite le droit ACCESS_FINE_LOCATION.
- L'accès à la position réseau nécessite le droit ACCESS_COARSE_LOCATION (inclus dans le précédent).

Position

- L'objet Location permet notamment de connaître :
  - La latitude, la longitude et l'altitude.
  - La précision, c'est à dire le rayon d'un cercle dans lequel l'utilisateur a 68% de chance de se trouver.
  - L'âge de cette information.
  - La vitesse et la direction, si celles-ci ont pu être calculées.
- Le calcul d'une distance entre deux Location peut se faire avec la méthode distanceTo().

Position

- De manière générale, il existe trois formats de représentation de la latitude/longitude :
  - Degrees:Minutes:Seconds : (49°30'00"N, 123°30'00"W).
    - 1 minute = 1/60 degré, 1 second = 1/3600 degré.
  - Degrees:Decimal Minutes : (49°30.0', -123°30.0'), (49d30.0m, -123d30.0').
  - Decimal Degrees : (49.5000°,-123.5000°).
    - 4 à 6 décimales.
    - Compris entre -180° et 180°.
- Android utilise le format Decimal Degree mais peut effectuer des conversions :
  - Location.FORMAT_DEGREES : [+-]DDD.DDDDD.
  - Location.FORMAT_MINUTES : [+-]DDD:MM.MMMMM
  - Location.FORMAT_SECONDS : DDD:MM:SS.SSSSS

```
double degree = Location.convert("123:30:05.5");
String minutes = Location.convert(degree, Location.FORMAT_MINUTES);
```

Position

- Il est possible d'utiliser des receivers à la place du LocationListener.
  - LocationManager.KEY_LOCATION_CHANGED : nouvelle position.
  - LocationManager.KEY_PROVIDER_ENABLED : booléen indiquant que le provider a été activé ou non.
  - LocationManager.KEY_STATUS_CHANGED : équivalent à onStatusChanged.

```
BroadcastReceiver receiver = new BroadcastReceiver()
{
     @Override
     public void onReceive(Context context, Intent intent)
     {
          Location location = (Location) intent.getExtras().get(LocationManager.KEY_LOCATION_CHANGED);
          ...
     }
};

IntentFilter filter = new IntentFilter("fr.inria.myapp.location");
registerReceiver(receiver, filter);

Intent intent = new Intent("fr.inria.myapp.location");
PendingIntent pending = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
locMgr.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 10000, 300, pending);
```

Position

- Lever une alerte de proximité avec un broadcast.

```
BroadcastReceiver receiver = new BroadcastReceiver()
{
     @Override
     public void onReceive(Context context, Intent intent)
     {
          boolean entered = intent.getExtras().getBoolean(LocationManager.KEY_PROXIMITY_ENTERING);
          ...
     }
};

IntentFilter filter = new IntentFilter("fr.inria.myapp.location.proximity.alert1");
registerReceiver(receiver, filter);

Intent intent = new Intent("fr.inria.myapp.location.proximity.alert1");
PendingIntent pending = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);

// latitude, longitude, radius, lifetime (-1 = infinite)
locMgr.addProximityAlert(48.808d, 2.134d, 200, 3600000, pending);
```

Position

- Obtenir des informations sur l'état du GPS, ainsi que sur les différents satellites (azimuth, élévation, signal to noise ratio, almanachs et éphèmérides à jours).

```
locMgr.addGpsStatusListener(new GpsStatus.Listener()
{
    @Override
    public void onGpsStatusChanged(int event)
    {
        // GPS_EVENT_FIRST_FIX, GPS_EVENT_SATELLITE_STATUS, GPS_EVENT_STARTED, GPS_EVENT_STOPPED
        if (event == GpsStatus.GPS_EVENT_SATELLITE_STATUS)
        {
            Iterable<GpsSatellite> sats = locMgr.getGpsStatus().getSatellites();
            for (GpsSatellite sat : sats)
            {
                ...
            }
        }
    }
});
```

Position

- Si l'API est simple, sa mauvaise utilisation peut rapidement drainer l'énergie de la batterie et dégrader l'expérience utilisateur.
  - Récupérer la position en cache pour ne pas faire attendre l'utilisateur.
  - Utiliser le PASSIVE_PROVIDER lorsque l'application passe en background (onPause).
  - Analyser les besoins en précision de votre application et sélectionner le provider (GPS ou réseau) en fonction du besoin.

POSITION

- Récupérer la position en cache pour ne pas faire attendre l'utilisateur lors du premier démarrage de l'application.

```
Location bestLocation = null;
List<String> providers = locMgr.getAllProviders();
for (String provider : providers)
{
    Location location = locMgr.getLastKnownLocation(provider);
    if (location != null)
    {
        if (isBetterLocation(location, bestLocation))
            bestLocation = location;
    }
}
```

- Cette fonction isBetterLocation() peut aussi servir ensuite à ne sélectionner que les meilleures mises à jour de position.

POSITION

```
protected boolean isBetterLocation(Location location, Location bestLocation)
{
    if (bestLocation == null) // a new location is always better than no location
        return true;

    // check whether the new location fix is newer or older
    long timeDelta = location.getTime() - bestLocation.getTime();
    boolean isSignificantlyNewer = timeDelta > TIME_THRESHOLD;
    boolean isSignificantlyOlder = timeDelta < -TIME_THRESHOLD;
    boolean isNewer = timeDelta > 0;

    if (isSignificantlyNewer)
        return true;
    else if (isSignificantlyOlder)
        return false;

    // check whether the new location fix is more or less accurate
    int accuracyDelta = (int) (location.getAccuracy() - bestLocation.getAccuracy());
    boolean isLessAccurate = accuracyDelta > 0;
    boolean isMoreAccurate = accuracyDelta < 0;
    boolean isSignificantlyLessAccurate = accuracyDelta > 200;

    boolean isFromSameProvider = isSameProvider(location.getProvider(), bestLocation.getProvider());

    if (isMoreAccurate)
        return true;
    else if (isNewer && !isLessAccurate)
        return true;
    else if (isNewer && !isSignificantlyLessAccurate && isFromSameProvider)
        return true;

    return false;
}
```

```
private boolean isSameProvider(String p1, String p2)
{
    if (p1 == null)
        return p2 == null;
    return p1.equals(p2);
}
```

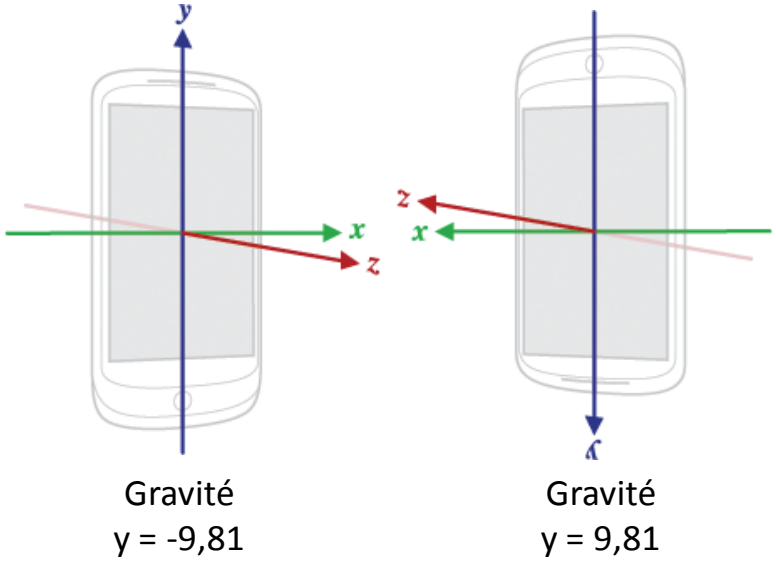
CAPTEURS DE MOUVEMENTS

Capteurs de mouvement et de position

- Capteurs de mouvement :
  - TYPE_ACCELEROMETER : un vecteur d'accélération en m/s^2.
  - TYPE_GRAVITY : un vecteur de gravité en m/s^2.
  - TYPE_GYROSCOPE : un vecteur de rotation en rad/s.
  - TYPE_LINEAR_ACCELERATION : un vecteur d'accélération mais sans la gravité.
  - TYPE_ROTATION_VECTOR : un vecteur d'orientation.
- Capteurs de position locale :
  - TYPE_PROXIMITY : une distance de l'objet le plus proche, en cm.
  - TYPE_MAGNETIC_FIELD : un vecteur de puissance du champ magnétique, en µT

Coordonnées

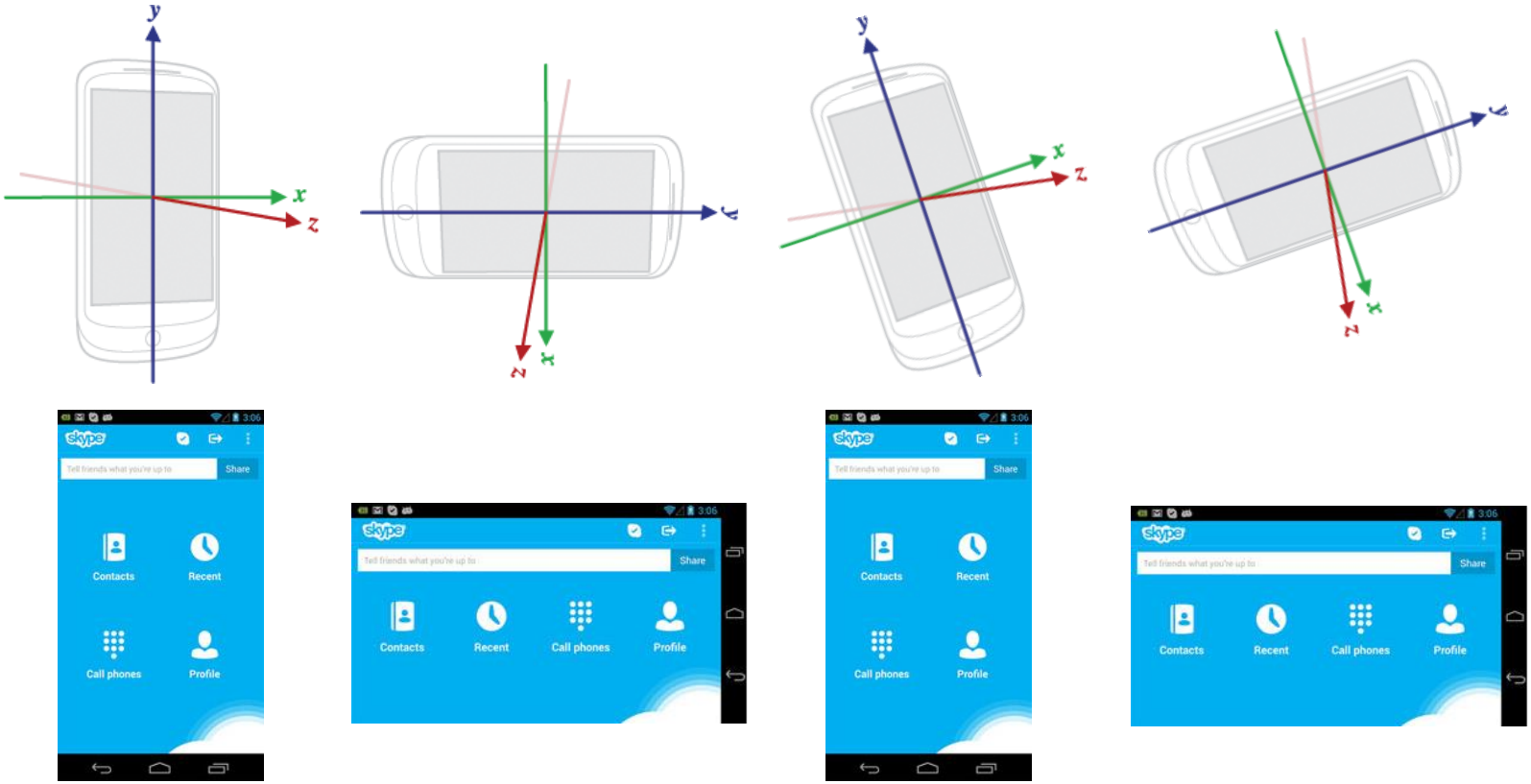
- La plupart des capteurs utilisent un système de coordonnées relatif à l'orientation du téléphone dans l'espace :
  - Acceleration sensor.
  - Gravity sensor.
  - Gyroscope.
  - Linear acceleration sensor.
  - Geomagnetic field sensor.



- Cependant, certains capteurs produisent des valeurs dans un système de coordonnées absolues (planétaires) :
  - Rotation vector sensor.

Orientation de l'application

- En premier lieu, il est nécessaire de détecter l'orientation de l'affichage. D'autant plus que certains appareils (tablettes) sont naturellement en paysage.



Orientation de l'application

```
getWindowManager().getDefaultDisplay().getRotation()
```

- Permet de déterminer quelle est l'orientation de l'application (indépendamment de l'orientation réelle du device) :
  - Surface.ROTATION_0
  - Surface.ROTATION_90
  - Surface.ROTATION_180
  - Surface.ROTATION_270
- Connaitre l'orientation permet de gérer les coordonnées du système relatif, indépendamment de l'orientation de l'application (pratique, par exemple, pour une application de réalité augmentée).

Passer aux coordonnées absolues

- Il est possible de passer les vecteurs du système de coordonnées relatives à un système de coordonnées absolues (planétaires).
- Pour se faire, on va récupérer une matrice de rotation qui servira à passer d'un système à l'autre au moyen d'une simple multiplication (comme dans OpenGL).
- Cette matrice de rotation permettra aussi de calculer l'orientation réelle de l'appareil (et pas uniquement de l'application).

Passer aux coordonnées absolues

- Pour construire la matrice de rotation, le système a besoin de se positionner par rapport au nord et par rapport à la gravité.
- Pour cela, on peut utiliser les capteurs TYPE_GRAVITY et TYPE_MAGNETIC_FIELD.
- Pour des raisons pratiques il est possible de demander la production d'une matrice 4x4, qui pourra directement être utilisée dans OpenGL :

```
float   float   float   0
float   float   float   0
float   float   float   0
0       0       0       1
```

PASSER AUX COORDONNÉES ABSOLUES

```
private float[] gravityValues;
private float[] magneticValues;
...
sensorMgr.registerListener(this, accelerometerSensor, SensorManager.SENSOR_DELAY_NORMAL);
sensorMgr.registerListener(this, gravitySensor, SensorManager.SENSOR_DELAY_NORMAL);
sensorMgr.registerListener(this, magneticSensor, SensorManager.SENSOR_DELAY_NORMAL);
...
@Override
public void onSensorChanged(SensorEvent event)
{
    if(event.sensor == gravitySensor)
    {
        gravityValues = new float[] { event.values[0], event.values[1], event.values[2] };
    }
    else if(event.sensor == magneticSensor)
    {
        magneticValues = new float[] { event.values[0], event.values[1], event.values[2] };
    }
    else if(event.sensor == accelerometerSensor && gravityValues != null && magneticValues != null)
    {
        float[] Rm = new float[16]; // rotation matrix
        float[] I = new float[16]; // inclination matrix

        SensorManager.getRotationMatrix(Rm, I, gravityValues, magneticValues);
        Matrix.invertM(Rm, 0, Rm, 0);

        float[] acceleration = new float[] { event.values[0], event.values[1], event.values[2], 0 };
        float[] result = new float[4];
        Matrix.multiplyMV(result, 0, Rm, 0, acceleration, 0);
    }
}
```

Calcul à partir de la rotation

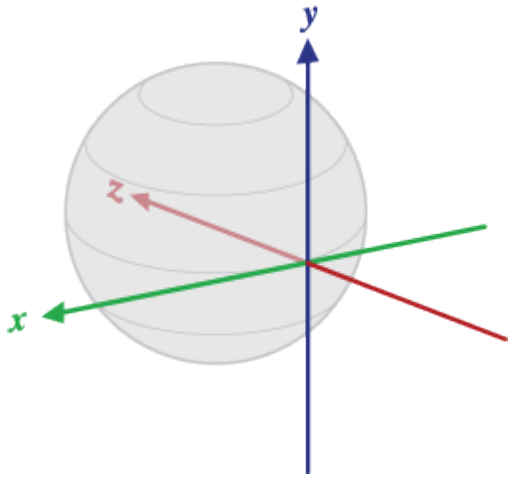
- A partir d'un vecteur de rotation (obtenue depuis le capteur TYPE_ROTATION_VECTOR), le calcul de la matrice de rotation est simplifié.

```
private float[] rotationVector;
...
sensorMgr.registerListener(this, rotationSensor, SensorManager.SENSOR_DELAY_NORMAL);
...
@Override
public void onSensorChanged(SensorEvent event)
{
    if(event.sensor == rotationSensor)
    {
        rotationVector = new float[] { event.values[0], event.values[1], event.values[2] };
    }
    else if(event.sensor == accelerometerSensor && rotationVector != null)
    {
        float[] Rm = new float[16]; // rotation matrix

        SensorManager.getRotationMatrixFromVector(Rm, rotationVector);
        Matrix.invertM(Rm, 0, Rm, 0);
        ...
    }
}
```

Déterminer l'orientation réelle

- La matrice de rotation permet de calculer l'orientation réelle de l'appareil :

```
// orientation[0]: azimuth, rotation around the Z axis.
// orientation[1]: pitch, rotation around the X axis.
// orientation[2]: roll, rotation around the Y axis.

float[] orientation = new float[3];
SensorManager.getOrientation(Rm, orientation);
```



Réorientation de la matrice

- Les axes de la matrice de rotation peuvent être réorientés au moyen de remapCoordinateSystem(), qui prend pour paramètres :
  - La matrice de rotation de départ.
  - L'axe sur lequel sera mappé l'axe X de l'appareil.
  - L'axe sur lequel sera mappé l'axe Y de l'appareil.
  - La matrice de sortie.
- Les différents axes sont AXIS_X, AXIS_Y, AXIS_Z, AXIS_MINUS_X, AXIS_MINUS_Y, AXIS_MINUS_Z.
- Très utile pour réorienter la matrice lorsque l'application est orienté (e.g., Surface.ROTATION_90).

Utiliser le capteur de proximité

- Le capteur de proximité permet de détecter la distance entre l'appareil et l'objet le plus proche (par exemple, la tête de l'utilisateur lors d'un appel).
- La plupart des capteurs de proximité retournent une valeur en cm.
- Cependant certains appareils ne retournent que deux valeurs équivalentes à "proche" ou "éloigné".
  - Utiliser getMaximumRange() pour connaître la valeur correspondant à "éloigné".

UTILISER LE CAPTEUR DE PROXIMITÉ

```
public class SensorActivity extends Activity implements SensorEventListener
{
     private SensorManager sensorMgr;
     private Sensor proximitySensor;

     @Override
     protected void onCreate(Bundle savedInstanceState)
     {
          ...
          sensorMgr = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
          proximitySensor = sensorMgr.getDefaultSensor(Sensor.TYPE_PROXIMITY);
     }
     @Override
     protected void onResume()
     {
          super.onResume();
          sensorMgr.registerListener(this, proximitySensor, SensorManager.SENSOR_DELAY_NORMAL);
     }
     @Override
     protected void onPause()
     {
          super.onPause();
          sensorMgr.unregisterListener(this);
     }
     @Override
     public void onSensorChanged(SensorEvent event)
     {
          float distance = event.values[0];
          boolean isFar = (distance == myProximitySensor.getMaximumRange() || distance > myThreshold);
          ...
     }
     @Override
     public void onAccuracyChanged(Sensor sensor, int accuracy)
     {
     }
}
```

CAPTEURS MULTIMÉDIA