Design and Development of Mobile Applications

Elias De Coninck - Pieter Simoens

Lab 3: Motion and Position sensors

Goal

The goal of this session is to implement an Activity that rotates a cube that is rendered with OpenGL by calculating the rotation matrix from the accelerometer, gyroscope and compass sensor readings.

The OpenGL ES APIs provided by the Android framework offer a set of tools for displaying high-end, animated graphics that benefit from the acceleration of graphics processing units (GPUs) provided on many Android devices.

Preparation

- Clone the **Labo-3** application into your workspace
  Use your UGent username and password.

  ```
  git clone https://github.ugent.be/OOMO/Labo-3
  ```

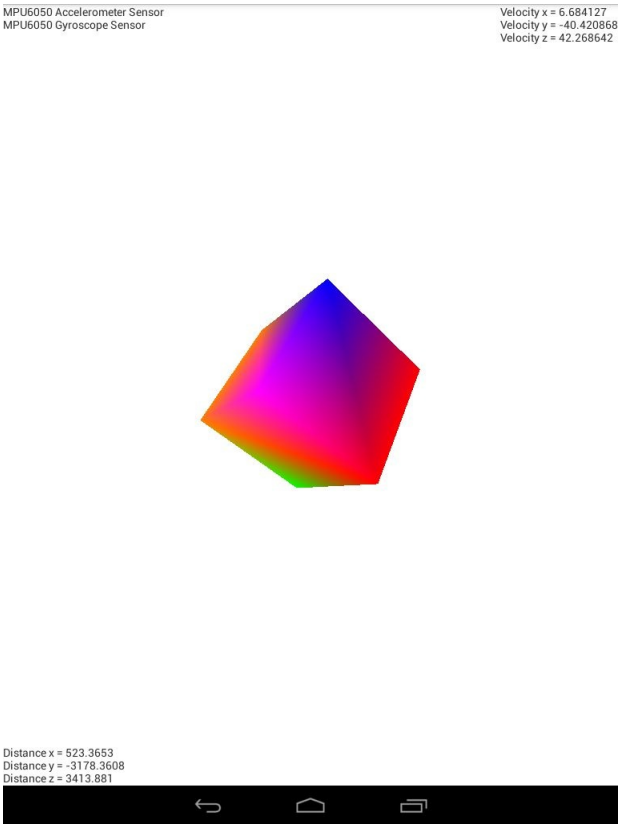
- Import the code as "Existing Android Project" into Eclipse.

Note on the emulator

The emulator does not support the emulation of hardware sensor readings. For this assignment, you can only work on a real device. Form a team of 2 students for this assignment.

Main Activity that uses GLSurfaceView

The goal is to create an Activity with the layout as shown in the figure:



- Create a new blank activity and add it as launcher activity to the AndroidManifest (New -> Other -> Android Activity).
- Create or update the XML layout file of your application as follows:
  - Use a RelativeLayout as root ViewGroup.
  - Add 3 TextView objects to the layout and give them an id (@+id/example).
  - The TextView in the top left corner will display sensor information.
  - The TextView in the top right corner will display velocity information.
  - The TextView in the bottom left corner will display distance information.
- Add a GLSurfaceView object to the RelativeLayout dynamically in the main activity. This is a special View container for objects drawn with OpenGL. Keep in mind that a RelativeLayout will stack views.
- In *onCreate*:
  - make sure you set the correct window feature and flag to make the Activity display in fullscreen.
  - set the content view of this activity to the XML layout file.
  - create in-memory references to the TextView objects via the R class.
  - set the renderer for the glSurfaceView by creating a new OpenGLRenderer, using the current Activity as Tracker argument for the constructor. This renderer will call the *getRotationMatrix* method and use the rotation matrix to display the cube.

    - The glSurfaceView should only be rendered when the rotation matrix was updated. Therefore, set the renderMode of the glSurfaceView to `RENDERMODE_WHEN_DIRTY`.

- Call the onPause() and onResume() methods of the glSurfaceView object in the onPause() and onResume() method of your activity.

- Make your Activity implement the Tracker interface
    - The method getRotationMatrix will return the current rotation matrix. Make sure you initialize the rotation matrix to a 4x4 identity matrix (what is the most appropriate lifecycle method to do this?). You can use a method from the android.opengl.Matrix class.

List all sensors in top left TextView

- Get a reference to the SensorManager and query it for all types of sensors (not only the motion sensors).

- Append each sensor's name to the top left TextView.

Rotate cube by reading from the accelerometer

- Update the AndroidManifest to use the accelerometer.

- Register the activity as listener for the default sensor of type `TYPE_ACCELEROMETER`.

    - Choose a good delay

    - What are the most appropriate lifecycle callback methods to register and unregister as listener?

- Implement the SensorEventListener interface
    - In *onSensorChanged*, check if the event originates from a sensor type `TYPE_ACCELEROMETER` Such events will contain a float vector in *event.values* (1x3).

    - Calculate the rotation matrix from *event.values* with the method *SensorManager.getRotationMatrix*. This function also needs a geomagnetic vector as input. For now, define a dummy value for this vector. Note that (0, 0, 0) is not a good value.

    - Make sure you request rendering of the glSurfaceView each time you have a new rotation matrix available.

At this point you should be able to launch the application on your device, and rotate the cube in the default device orientation. Unfortunately, this only works in the default For phones, when rotating to landscape mode the orientation of the device is no longer equal to the orientation of the cube.

When not in default mode, the device coordinate system (used by the SensorManager) is no longer aligned to the screen coordinate system used by OpenGL. We must recalculate the rotation matrix by remapping the coordinate system.

- Get the current rotation of the device

- If the rotation is equal to `Surface.ROTATION_0`, no further action is required.

- If the rotation is equal to `Surface.ROTATION_90`, `ROTATION_180`, `ROTATION_270`, you should call *remapCoordinateSystem* to recalculate the rotation matrix.

Rotating the device will now remap the coordinate system and rotation should feel natural. The cube will rotate with the device, but has a lot of jitter.

- Add a low-pass filter to minimize the jitter.

Improve accuracy [if time permits]

The available devices have different sensors.

Devices with MAGNETIC_FIELD sensor: use readings to calculate rotation matrix.

- Register your Activity as listener for the default sensor of type `TYPE_MAGNETIC_FIELD`. Do not forget to unregister when appropriate.
- Instead of the dummy vector you used before, now use the actual readings from the geomagnetic vector.
- Add a low-pass filter to minimize jitter.

Devices with GYROSCOPE sensor: calculate the rotation matrix from gyroscope instead of accelerometer.

- Register your Activity as listener for the default sensor of type `TYPE_GYROSCOPE`. Do not forget to unregister when appropriate.
- Calculate the rotation vector from the rotation angles in event.values
  - Use SensorManager.getRotationMatrixFromVector to get a delta rotation matrix (see Android docs on the gyroscope).
  - Use Matrix.multiplyMM to multiply the current rotation matrix with the delta rotation matrix. Hint: multiplyMM does not calculate the matrix multiplication in place so you'll need to clone the original rotation matrix.
- Account for device orientation changes like you did for the accelerometer.

Calculate the velocity from accelerometer [if time permits]

- Add a high-pass filter to the acceleration vector to remove the gravity contributions
  This will give us the linear acceleration.
- Calculate the velocity from the linear acceleration.
- Update the top right TextView with the current velocity.

Submit assignment

Rename `MainActivity.java` to `NAME_OOMO_LABO_3.java` and submit to the Minerva Dropbox. Make sure to send it to 'Cursusbeheerders'.