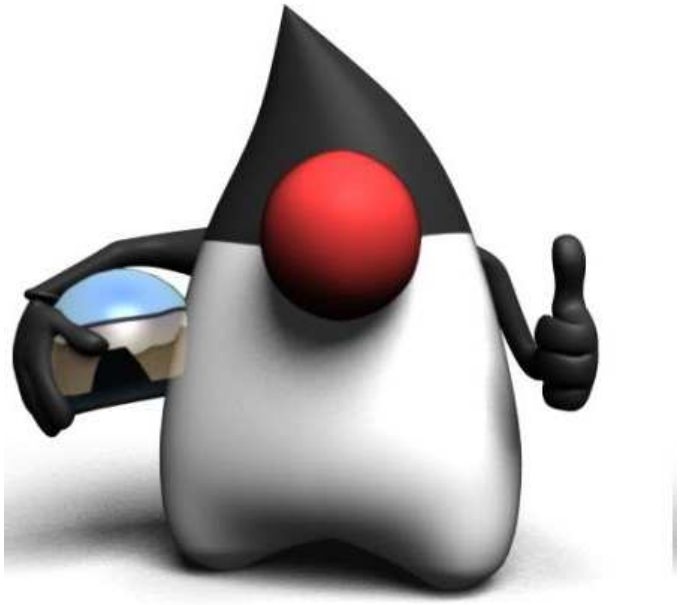
Android
-
Orientácia, rotácie

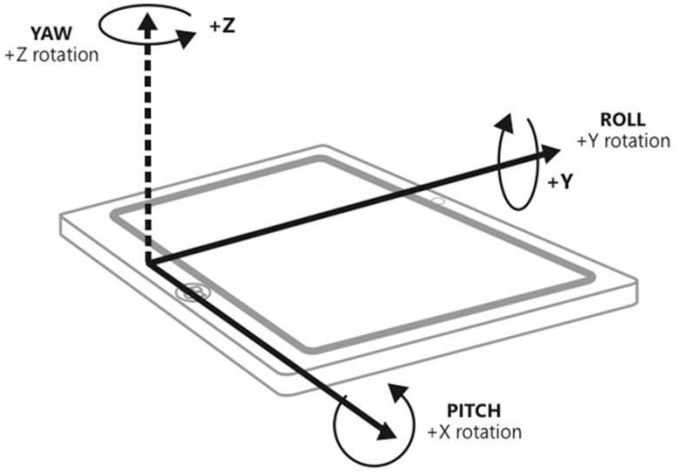


Peter Borovanský
KAI, I-18

borovan 'at' ii.fmph.uniba.sk

Rotácia zariadenia
Rotácia objektu
Rotácia displaya
Matica rotácie
Quaterniony

Orientácia



Ako zistiť natočenie zariadenia ?

správna otázka: vzhľadom na čo... Zem ? Sever ?

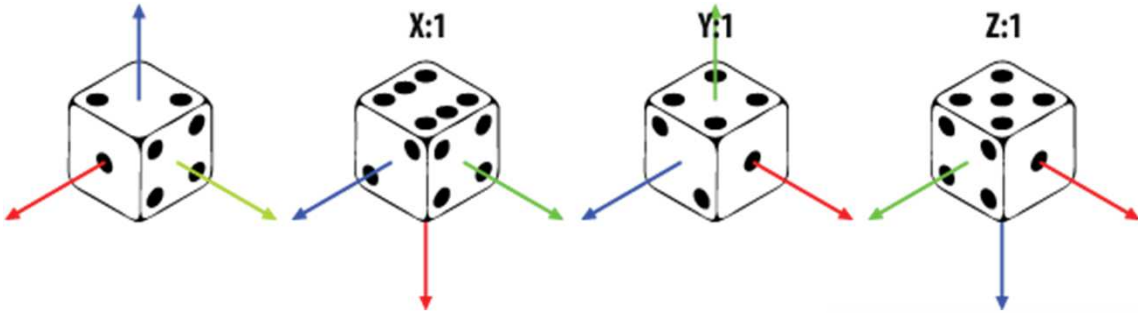
- akcelerometer zo zrýchlenia, t.j. gravitácie [m/s^2]
  - HW akcelerometer – šum a problém filtrácie (minule)
  - SW lineárny akcelerometer ignoruje gravitáciu
  - gravity senzor - SW fúzia HW senzorov (Accelero/Gyro/Magneto)
    - odstraňuje šum, či
    - faktor zemskej gravitácie,
    - implementácia je špecifická od modelu zariadenia
- gyroskop [rad/s]
  - meria uhlovú rýchlosť, nie natočenie => problém integrácie, vzniká chyba
- orientačný senzor (SW fúzia Accelo+Magneto)

  zastaralý spôsob, ktorý sa (google) nepodporuje
- kombináciou akcelerometra a magnetometer (kompasu)

Ako adekvátne zobraziť natočenie nejakého objektu na display?

správna otázka: Čo je adekvátne ?...

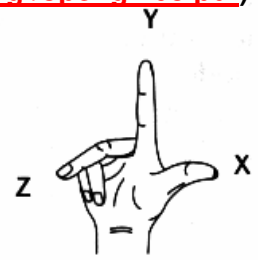
Rotácia
(úvod)



(viac: http://fractal.dam.fmph.uniba.sk/~samuelcik/opengl/opengl_03.pdf)

Z grafiky poznáme OpenGL:

- `gl.glRotatef(90f, 1.0f, 0.0f, 0.0f);`
- `gl.glRotatef(90f, 0.0f, 1.0f, 0.0f);`
- `gl.glRotatef(90f, 0.0f, 0.0f, 1.0f);`



Inverzné otočenie k `glRotatef(angle, x, y, z)`

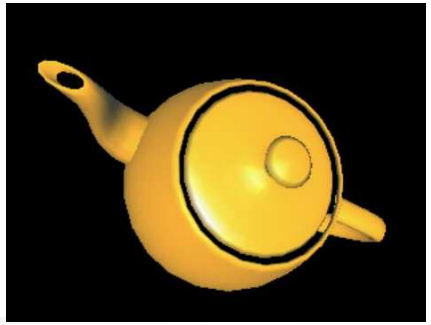
- je okolo opačnej osi `glRotatef(angle, -x, -y, -z)`, alebo
- o opačný uhol `glRotatef(-angle, x, y, z)`

Inverzné otočenie k postupnosti otočení je otočená postupnosť inverzných otoč:

- `gl.glRotatef(-90f, 0.0f, 0.0f, 1.0f);`
- `gl.glRotatef(-90f, 0.0f, 1.0f, 0.0f);`
- `gl.glRotatef(-90f, 1.0f, 0.0f, 0.0f);`

http://blog.uncle.se/2012/02/opengl-es-tutorial-for-android-part-iii-transformations/

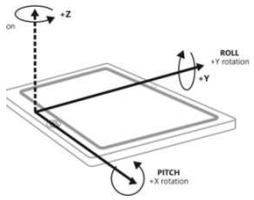
Project:Sensor.TeaPot_ORIENTATION.zip



TYPE_ORIENTATION

(deprecated...)

```
public void onSensorChanged(SensorEvent event) {
float[] sensorData = new float[3]; // orientácia zariadenia
    sensorData[0] = (float) event.values[0]; // azimut
    sensorData[1] = (float) event.values[1]; // pitch
    sensorData[2] = (float) event.values[2]; // roll
```



Ako natočiť predmet, aby jeho zobrazenie zodpovedalo zariadeniu ?

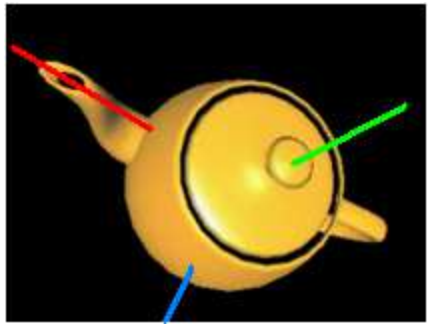
```
gl.glRotatef(sensorValues[0], 0, 0, 1);          // azimut
gl.glRotatef(-sensorValues[1], 0, 1, 0);         // pitch
gl.glRotatef(sensorValues[2]+ 90.0f , 1, 0, 0); // roll
```

Prečo je tam:

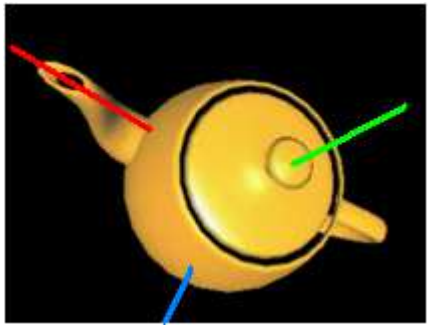
`-sensorValues[1]?`

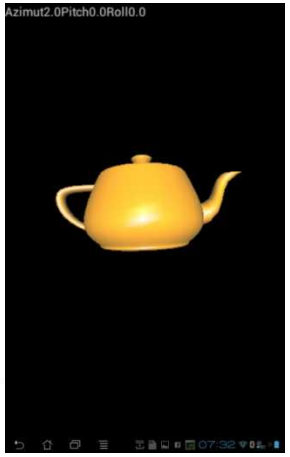
`+90.0f?`

čo ak zmeníme poradie

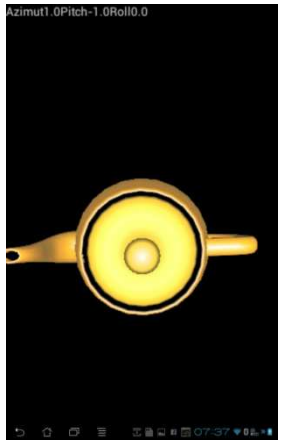


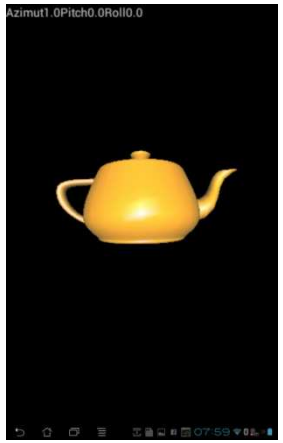
TeaPot kvíz






```
gl.glRotatef(sensorValues[0], 0, 0, 1); azimut
gl.glRotatef(-sensorValues[1], 0, 1, 0); pitch
gl.glRotatef(sensorValues[2] , 1, 0, 0); roll
------------------------------
gl.glRotatef(sensorValues[0]+180.0f, 0, 0, 1);
gl.glRotatef(sensorValues[1], 0, 1, 0);
gl.glRotatef(-sensorValues[2]+ 90.0f , 1, 0, 0);
------------------------------
gl.glRotatef(sensorValues[0], 0, 0, 1);
gl.glRotatef(sensorValues[2] , 1, 0, 0);
gl.glRotatef(-sensorValues[1], 0, 1, 0);
```





Minimálne morálne ponaučenie:

- **android má podporu pre OpenGL**
- **výstup z ORIENTATION senzora je intuitívny**
- **a *nejaký ogl-čajník* nájdete napr. v TeaPot.zip...**

Project:Sensor.TeaPot.zip

Niektoré afinné 2D-zobrazenia

- Posunutie o vektor [a,b]:
  (zhodné zobrazenie, nemení veľkosť)

$$(x, y) \rightarrow (x+a, y+b)$$

- Nafúknutie a-krát v smere x,
  b-krát v smere y

$$(x, y) \rightarrow (ax, by)$$

  - Rovnoľahlosť, ak a=b (podobné zobrazenie, nemení uhly)
  - Osová symetria: (zhodné zobrazenie, nemení veľkosť)
    podľa x, ak a=1, b=-1
    podľa y, ak a=-1, b=1
- Rotácia okolo [0,0] o uhol θ v smere hodinových ručičiek
  (zhodné zobrazenie, nemení veľkosť)

$$(x, y) \rightarrow (x \cos(\theta)+y \sin(\theta), -x \sin(\theta)+y \cos(\theta))$$

- Rotácia okolo [a,b] o uhol θ ?
  - posunutie [-a, -b] posunie stred otáčania do [0,0]
  - otoč enie o θ
  - posunutie [a, b] posunie počiatok [0,0] do stredu otáčania [a,b]

Matice 2D-zobrazení

- Symetria podľa osi y

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Symetria podľa osi x

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Symetria podľa uhlopriečky y=x

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Rotácia okolo [0,0] o uhol θ (http://sk.wikipedia.org/wiki/Line%C3%A1rne_zobrazenie)

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta + y\sin\theta \\ -x\sin\theta + y\cos\theta \end{bmatrix}$$

- A posunutie o [a,b] ??? To nevieme napísať pomocou matice 2x2 :-(
  … preto vymysleli tzv. homogénne súradnice, a pridali jednu zložku

Homogénne 2D-súradnice

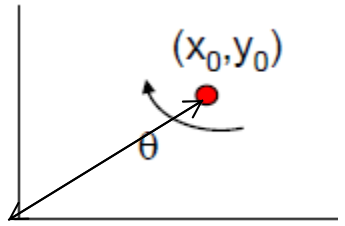
Pre 2D sú to trojice, pre 3D to budú štvorice $(x, y) \rightarrow (tx, ty, t)$

$$(X, Y, W) \rightarrow \left(\frac{X}{W}, \frac{Y}{W}\right)$$

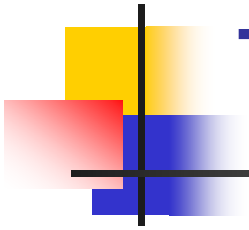
Posunutie o vektor [a,b]:

$$\begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + a \\ y + b \\ 1 \end{bmatrix}$$

Rotácia okolo [x_0, y_0] o uhol θ v smere hod.ručičiek:



$$\begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



To bolo v 2D

- V 3D pribudne jedna súradnica, ale princíp zostáva

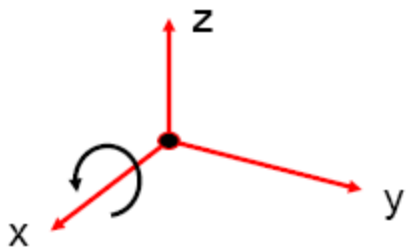
Eulerove uhly

http://www.cs.brandeis.edu/~cs155/Lecture_07_6.pdf

Každá rotácia sa dá poskladať z troch elementárnych rotácií o uhly okolo osí x,y,z

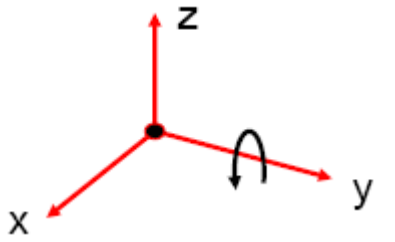
- Counterclockwise rotation about x-axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



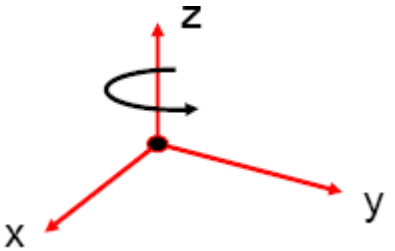
- Counterclockwise rotation about y-axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



- Counterclockwise rotation about z-axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

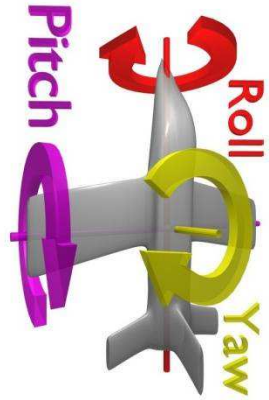


Každá rotácia je súčinom 3 matíc elementárnych rotácií o $\theta_x\theta_y\theta_z$ okolo x, y, z.

Ale násobenie matíc NIE JE komutatívne, preto aj pri otáčaní ZALEŽÍ na poradí osí

Problém Pitch±90

(Gimbal Lock)



Rotácia vyjadrená pomocou Eurlerových uhlov (azimut, pitch a roll) má problém, ak pitch je blízko ±90º,
t.j. ak naša raketa, auto, čajník ukazuje priamo do neba, či kolmo do zeme.

Vtedy sa azimut (yaw) a rotácia (roll) správajú rovnako.
Do tejto nestabilnej polohy sa dá rovnako dostať ako

- azimut(θ)+pitch 90, ale aj ako
- pitch 90+roll(θ).

Porovnanie RM, SLERP, Quat

Alternatívne techniky rotácie v 3D sú tzv.quaterninony.
Poučenie:

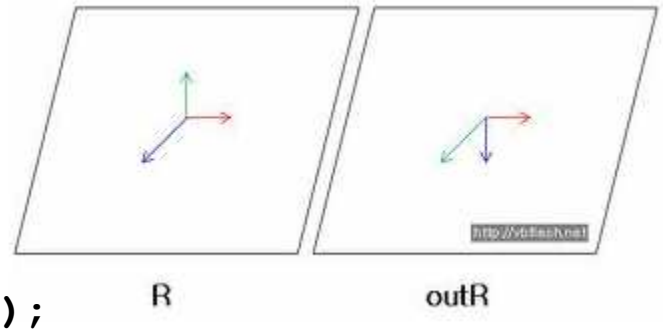
- ak budete programovať nejakú hru s možnosťou ísť Pitch±90º, tak vás quaterniony dobehnú...
- android a OpenGL má na to podporu :-)

detaily:http://www.youtube.com/watch?v=zc8b2Jo7mnohttp://www.youtube.com/watch?v=zc8b2Jo7mno

Kompas
(rotation matrix)

Project:SensorHorizont, Compass.zip

```
public void onSensorChanged(SensorEvent event) {
 if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
     accelo = event.values.clone();
 if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
     magnet= event.values.clone();
 float[] R = new float[9];              // rotatiom matrix 3x3
//float[] R = new float[16];            // rotatiom matrix 4x4
 SensorManager.getRotationMatrix(R, null, accelo, magnet);
 float[] outR = new float[9];           // new rotatiom matrix
 SensorManager.remapCoordinateSystem(R,
                     SensorManager.AXIS_X,
                     SensorManager.AXIS_Z,
                     outR);
 float[] values = new float[3];
 SensorManager.getOrientation(outR, values);
 azimut= (float) Math.toDegrees(values[0]); // radian->deg
 pitch = (float) Math.toDegrees(values[1]);
```



Project:Sensor.SensorSmallCompas.zip

Problémy s kompasom

toto u mňa NIEKEDY nespĺňa ani jeden kompas :-(

```
if (event.accuracy ==
   SensorManager.SENSOR_STATUS_UNRELIABLE) {
    return;
 }   // nechápem a nerozumiem, ale miera zúfalstva bola veľká
```

http://www.youtube.com/watch?v=sP3d00Hr14o

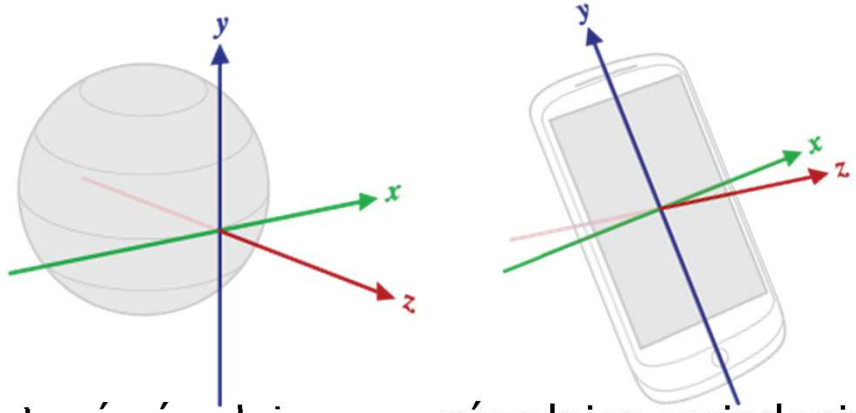
Ako si trochu pomôcť ?

```
if (uselPassFilter) {
   magnet[0] = lPass*event.values[0]+(1f-lPass)*magnet[0];
   magnet[1] = lPass*event.values[1]+(1f-lPass)*magnet[1];
   magnet[2] = lPass*event.values[2]+(1f-lPass)*magnet[2];
} else {
   magnet = event.values.clone();
}
```

Project:Sensor.SensorSmallCompas.zip

Čo je rotation matrix

```
SensorManager.getRotationMatrix(R, I, accelo, magnet);
```



svetové súradnice súradnice zariadenia

Matica rotácie je zobrazenie transformujúce súradnice zariadenia na svetové.

Matica rotácie ignoruje landscape/portrait natočenie displaya, to treba doriešiť...

Ak zariadenie leží vodorovne na chrbte, otočené y na sever, R je skoro jednotková.

```
 0.9996201,    0.0032320707,  -0.027374819          1  0  0
-0.004806483,  0.9983257,     -0.057644155          0  1  0
 0.027142672,  0.057753824,    0.9979618            0  0  1
```

Test: Ako vyzerá matica rotácie, ak mobil leží, smeruje na východ/západ/juh/nebo?

Test: riešenia

Východ: 0 1 0, t.j. X′ = y, Y′ = -x, Z′ = z
-1 0 0
0 0 1

Juh: -1 0 0, t.j. X′ = -x, Y′ = -y, Z′ = z
0 -1 0
0 0 1

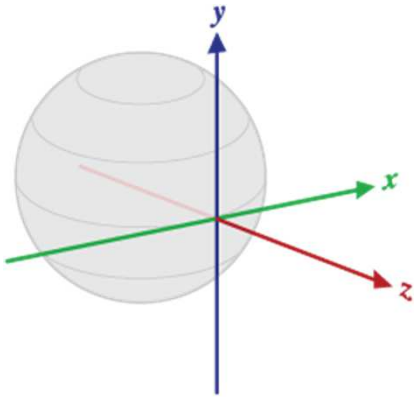
Západ: 0 -1 0, t.j. X′ = -y, Y′ = x, Z′ = z
1 0 0
0 0 1

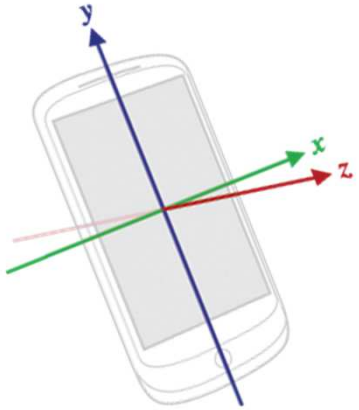
"nebo": (pozor na Pitch ±90º):
0 0 -1, t.j. X′ = -z, Y′ = y, Z′ = x
0 1 0
1 0 0

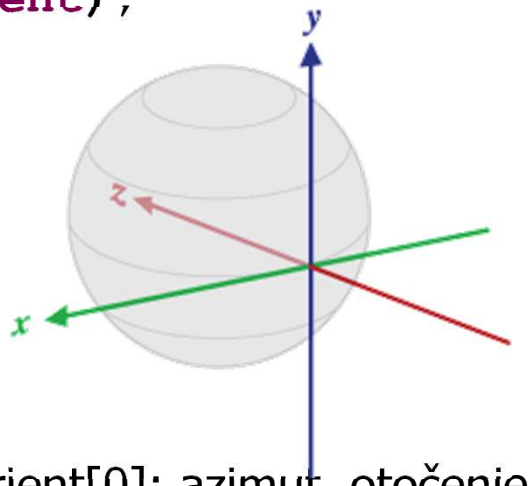
Hádanka: nakreslite poslednú pozíciu mobilu

Čo je getOrientation

```
SensorManager.getRotationMatrix(R, I, accelo, magnet);
SensorManager.getOrientation(R, Orient);
```







[-PI; PI] Orient[0]: azimut, otočenie okolo Z

[-PI/2; PI/2] Orient[1]: pitch, otočenie okolo X

[-PI; PI] Orient[2]: roll, otočenie okolo Y

Orientácia sa vzťahuje na uhly natočenia zariadenia z neutrálnej polohy, t.j. **na vodorovnom stole ležiac na chrbte a vrchom na sever**

Pozor: zastaralý TYPE_ORIENTATION senzor má iné rozsahy

Azimut: [0;360], pitch: [-180; 180], roll: [-90; 90]

A čo landscape/portrait display

AndroidManifest.xml:

- zakážeme ho:
  `android:screenOrientation="portrait„`
- povolíme všetky 4 orientácie display:
  `android:screenOrientation="fullSensor„`
- aby sa nám aplikácia neresetovala pri zmene zobrazenia:
  `android:configChanges="keyboardHidden|orientation|screenSize"`

```
public void onConfigurationChanged(Configuration newConfig)
    super.onConfigurationChanged(newConfig);
    mScreenRotation =    // mScreenRotation * 90 degrees
        getWindowManager().getDefaultDisplay().getRotation();
}
```

ORIENTATION_PORTRAIT, ORIENTATION_LANDSCAPE
LANDSCAPE_REVERSE, PORTRAIT_REVERSE

Project:Sensor.SensorSmallCompas.zip

Surface.Rotation
(azimut)

```
switch (mScreenRotation) {
   case Surface.ROTATION_0:
   break;
   case Surface.ROTATION_90:
        azimut+= Math.PI / 2f;
   break;
   case Surface.ROTATION_180:
        azimut= (float) (azimut > 0f ?
        (azimut- Math.PI) : (azimut+ Math.PI));
   break;
   case Surface.ROTATION_270:
        azimut-= Math.PI / 2;
   break;
}
```

bug (??) v prepínaní medzi 180<->0 a 270<->90

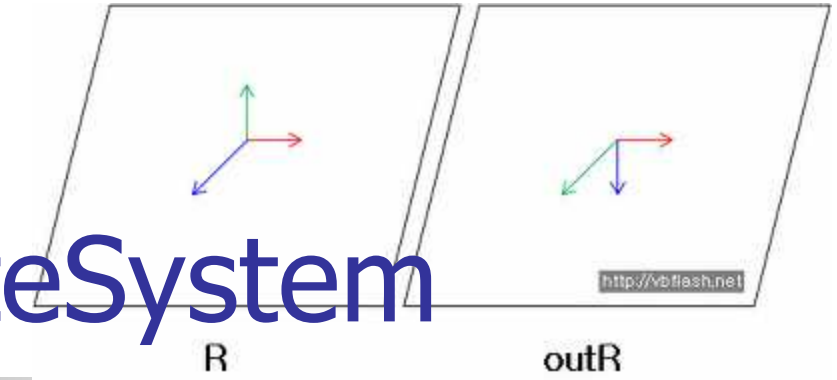
Project:Sensor.SensorSmallCompas.zip

Surface.Rotation

(akcelerometer)

```
public void onSensorChanged(SensorEvent event) {
  if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER){
    switch (mDisplay.getRotation()) {
        case Surface.ROTATION_0:
            mSensorX = event.values[0];          // x    1   0
            mSensorY = event.values[1]; break; // y    0   1
        case Surface.ROTATION_90:
            mSensorX = -event.values[1];         // -y   0 -1
            mSensorY = event.values[0]; break; // x    1   0
        case Surface.ROTATION_180:
            mSensorX = -event.values[0];         // -x  -1   0
            mSensorY = -event.values[1]; break;// -y   0 -1
        case Surface.ROTATION_270:
            mSensorX = event.values[1];          // y    0   1
            mSensorY = -event.values[0]; break;//-x   -1   0
```

remapCoordinateSystem



remapCoordinateSystem (float[] R, int X, int Y, float[] outR)

R matica rotácie z getRotationMatrix(float[], float[], float[], float[]),
X na ktorú svetovú os sa mapuje X-os zariadenia
Y na ktorú svetovú os sa mapuje Y-os zariadenia
outR transformovaná matica

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

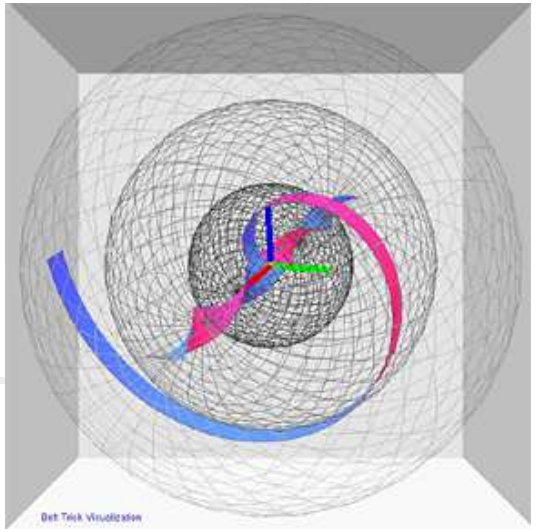
remapCoordinateSystem(R, SensorManager.AXIS_X,
SensorManager.AXIS_MINUS_Z, outR)

- zmení súradnicový systém tým, že nastaví nové osi X, Y (aj Z) na osi natočeného zariadenia AXIS_X a AXIS_MINUS_Z .

Ak používame kameru (napr. pre augumented reality)

- v portait mode: SensorManager.AXIS_X, SensorManager.AXIS_MINUS_Z.
  resp. SensorManager.AXIS_X, SensorManager.AXIS_Z.
- landscape mode: SensorManager.AXIS_Y, SensorManager.AXIS_MINUS_Z.

Quaterniony



Myšlienka za tým:
Každá rotácia v priestore sa dá vyjadriť ako rotácia
o uhol θ okolo nejakej osi <x*sin(θ/2), y*sin(θ/2), z*sin(θ/2)>

Fakt1:

Sensor.TYPE_ROTATION_VECTOR vracia „quaternion" ako 3-prvkové pole, t.j.
event.value = <x*sin(θ/2), y*sin(θ/2), z*sin(θ/2)>

Fakt2:

ak z toho chceme rotačnú maticu (viac v SensorRotationVector.zip), tak
SensorManager.getRotationMatrixFromVector(mRotationMatrix, event.values);

Fakt3:

ak to chceme správne otočiť, tak

```
gl.glRotatef(  (float)(2.0f*Math.acos(sensorValues[0])*180.0f / Math.PI),
                sensorValues[0], sensorValues[1], sensorValues[2]);
```

Project:Sensor.TeaPot_Quaternion.zip

Quaterniony do budúcna

Sensor.TYPE_ROTATION_VECTOR vracia

- values[0]: x*sin(θ/2)
- values[1]: y*sin(θ/2)
- values[2]: z*sin(θ/2)
- values[3]: cos(θ/2) - nebude treba arccos...
- values[4]: estimated heading Accuracy (in radians) (-1 if unavailable)

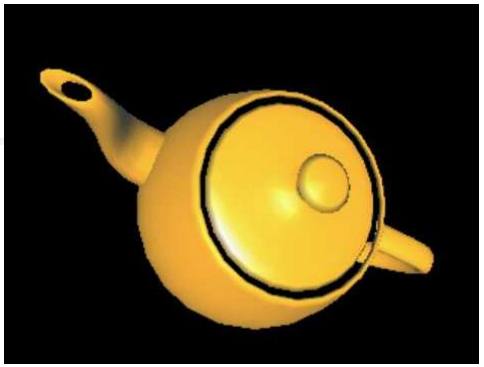
Ale pozor:

values[3], originally optional, will always be present from SDK Level 18 onwards.

values[4] is a new value that has been added in SDK Level 18.

Demá

Project:Sensor.TeaPot.zip
Project:Sensor.TeaPotRotationMatrix.zip
Project:Sensor.TeaPot_QUATERNION.zip

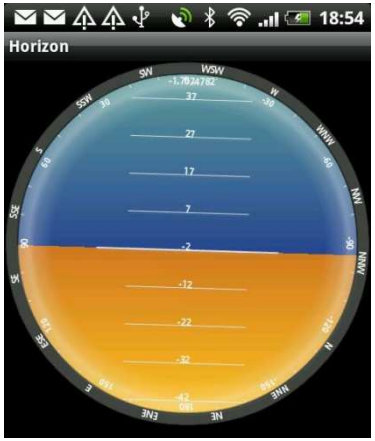


- Čajník
  - *TYPE_ORIENTATION*
  - *openGL._GLSurfaceView*
- Compass
  - *TYPE_ACCELEROMETER, TYPE_MAGNETIC_FIELD*
  - *plávajúci priemer, buff size 10, 100, ...*
- Horizont (vitruálny)
  - *TYPE_ACCELEROMETER, TYPE_MAGNETIC_FIELD*
  - Weighted smoothing

Project:Sensor.Compass.zip

Project:Sensor.Horizont.zip

Project:SensorRotationVector.zip



Screenshot