# Lehrerfortbildung: 3D Modellierung Werkstatt Java/Android mit OpenGL

Maximilian Schiedermeier

Universität Passau

29. Juni 2017

# Hallo

## Bevor wir Anfangen...

Sonstiges:
Sport&Natur, Musik,
Bücher, Fotos&Reisen

Studium:
09/2017: Master Inf. (Passau, Lyon)
Fachrichtung:
Software Engineering, Privacy,
Mobile Systeme

Berufliches:
HiWi, Embedded Systems Labs, Passau
WebDev, Atos Worldline, Lyon
HiWi, Exccess, IRIXYS
Grad. Res. Trainee, McGill, Montréal

# Hallo

## Ablauf

| | |
|---|---|
| 9:00 - 9:15 | Begrüßung |
| 9:45 - 11:00 | Eine minimale Pipeline |
| 11:00 - 11:20 | **Kaffee** |
| 11:20 - 12:30 | Automatisierter Szenenaufbau |
| 12:30 - 13:15 | **Mittagspause** |
| 13:15 - 14:30 | Animierte Bewegungen |
| 14:30 - 14:50 | **Kaffee** |
| 14:30 - 15:45 | Massenträgheit und Sensoren |

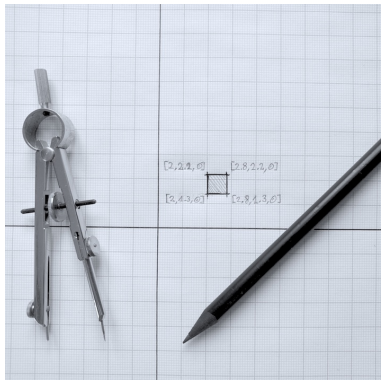
1. Minimale Pipeline
2. Automatisierter Szenenaufbau
3. Animierte Bewegungen
4. Goodies

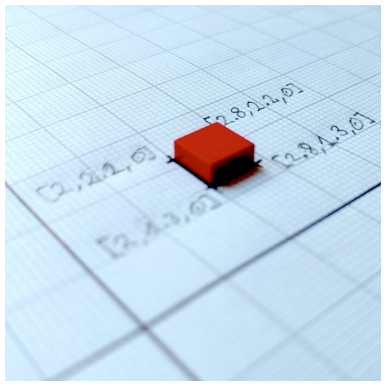
# Programm

# Minimale Pipeline

## Ein bisschen Theorie

Wie berechnet man eigentlich ein 3D Bild?
$\rightarrow$ Genau wie in der Realität:

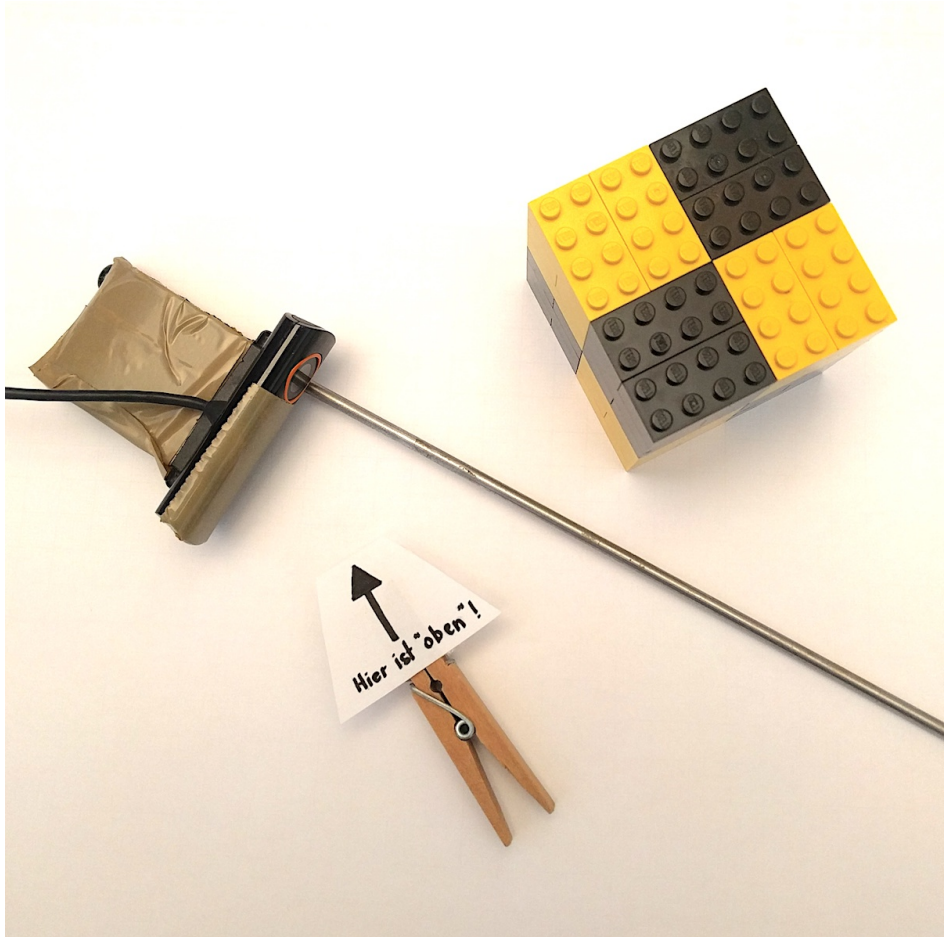




- Objekte platzieren
- Betrachtungsort und -Winkel wählen
- Foto machen

# Minimale Pipeline

## Demo!



# Minimale Pipeline

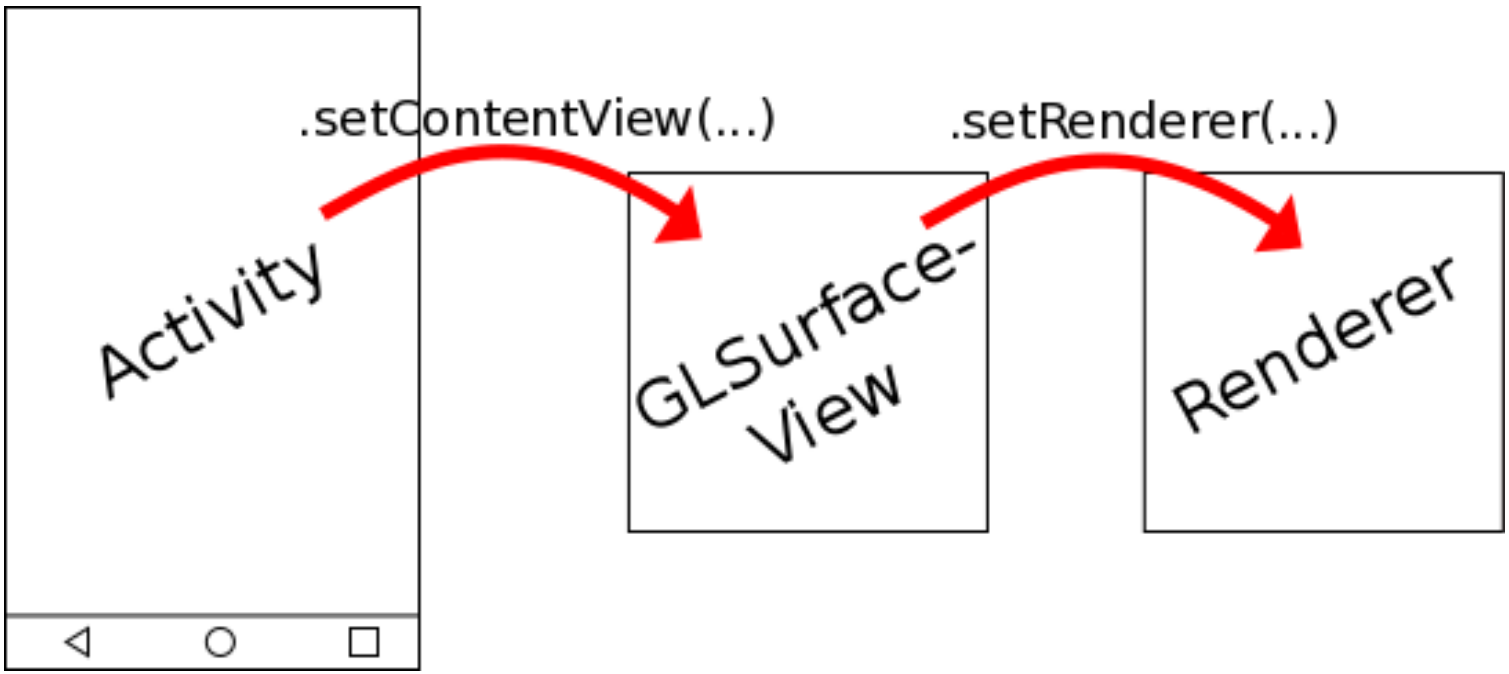
## Technische Umsetzung

Wie implementieren wir diese Pipeline auf Android?
$\rightarrow$Mit OpenGL:

- Native Bibliothek
- Werkzeuge für Modellierung
- Hardware-beschleunigt (GPU)

# Minimale Pipeline

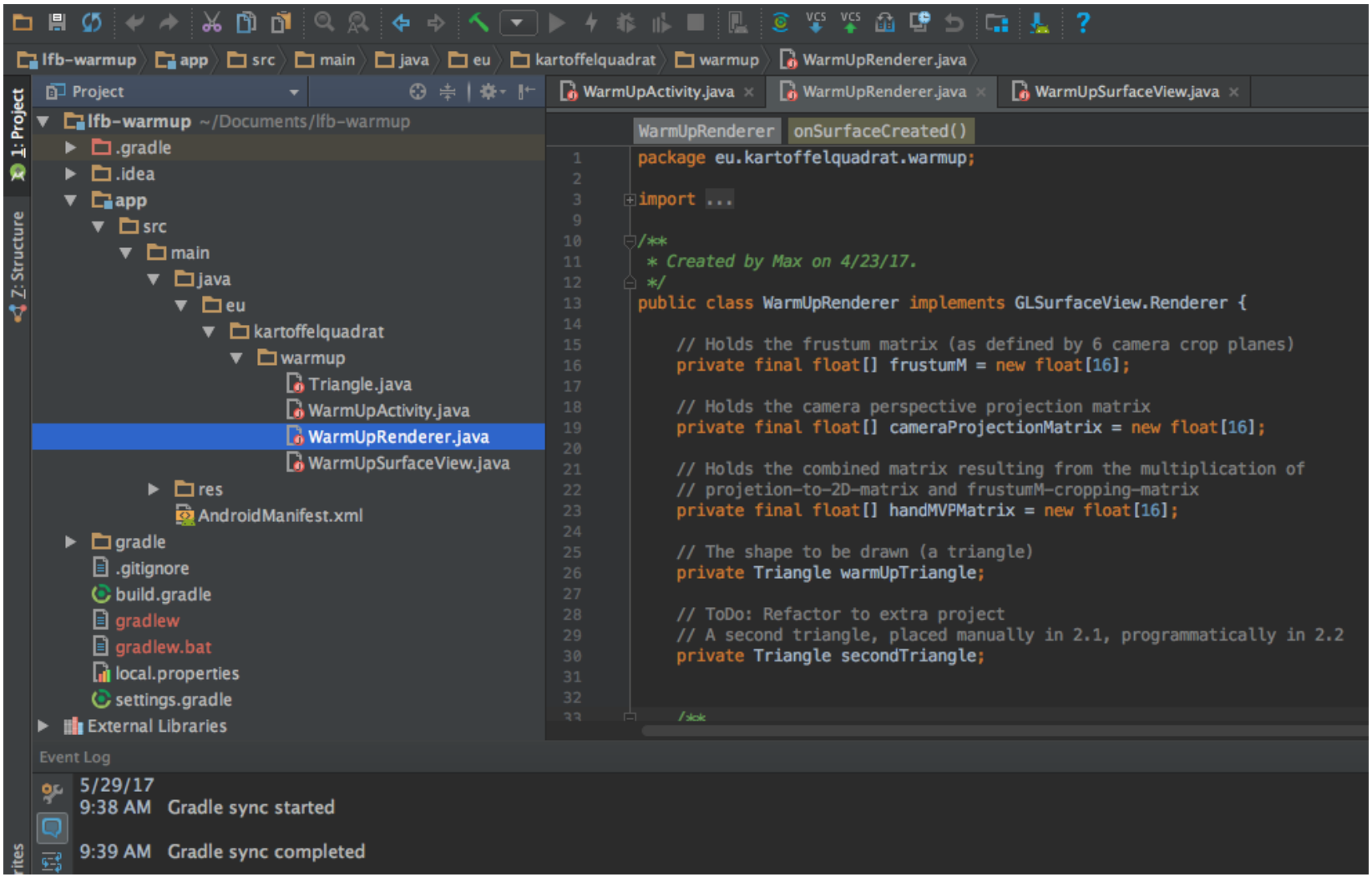
## In der Praxis




Anmerkung: Die komplette Pipeline *lebt* im Renderer.

# Minimale Pipeline

## Android Studio



# Minimale Pipeline

## An die Arbeit...

**Aufgabe 0**:

- Öffnen Sie das Projekt mit Android Studio:
  *File→Open...→AndroidStudioProjects/lfb-warmup*
- Öffnen Sie die Klasse *Renderer*

# Minimale Pipeline

## An die Arbeit...

**Aufgabe 1**:

- Identifizieren Sie die Pipeline ($\rightarrow$ *Renderer*)
  - Wo werden *Objekte platziert*?
  - Wo wird die *Kamera platziert / ausgerichtet*?
  - Wo wird die *Pipeline angewandt*?
- Was könnte die Anzeige des Programms sein?

# Minimale Pipeline

## An die Arbeit...

**Kurze Rekapitulation, zusammen**:

```
void onSurfaceCreated(GL10 gl10, EGLConfig eglConfig) {
 warmUpTriangle = new Triangle(-0.5f, -0.3f, 0.5f, -0.3f, 0.0f, 0.58f, 0f, 1f, 0f);
...}

void onFrawFrame(GL10 gl10) {
 Matrix.setLookAtM(cameraPositionMatrix, 0, 0, 0, -6, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
 Matrix.multiplyMM(handMVPMatrix, 0, frustumM, 0, cameraPositionMatrix, 0);

 warmUpTriangle.draw(handMVPMatrix); }
```

**Aufgabe 2**:

- Überprüfen Sie ihre Vermutung:
  - Führen Sie einen Gradle-SYNC aus
  - Koppeln sie ein Android per: `adb devices`
  - Starten Sie das Projekt
- Ändern Sie die Dreieck-Farbe
- Ändern Sie die Kameraperspektive $\rightarrow$ Das Dreieck soll nun in Schrägansicht angezeigt werden.
- Platzieren Sie *daneben* ein zweites Dreieck.
- Ändern sie die Initialisierung des *ersten* Dreiecks, so das es *kopfüber* steht.

# Programm

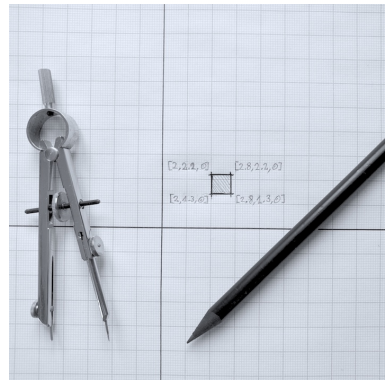
1. Minimale Pipeline
2. Automatisierter Szenenaufbau
3. Animierte Bewegungen
4. Goodies

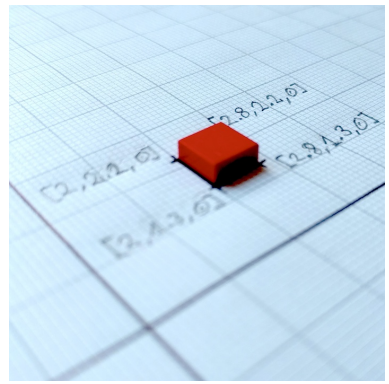
# Automatisierter Szenenaufbau

## Kurze Erinnerung...

Unsere mini-Pipeline hat 3 Schritte:

- Objekte platzieren ← *Hierum geht's*
- Betrachtungsort und -Winkel wählen
- Foto machen





# Automatisierter Szenenaufbau

## Kurze Erinnerung...

Was haben wir gemacht?

- Platzieren eines Dreiecks, bislang:

```
void onSurfaceCreated(GL10 gl10, EGLConfig eglConfig) {
 warmUpTriangle = new Triangle(-0.5f, -0.3f, 0.5f, -0.3f, 0.0f, 0.58f, 0f, 1f, 0f);

...}
```

- Andere Position oder Drehung $\rightarrow$ Wahl anderer Eck-Koordinaten.
- Berechnung: Entweder manuell oder im Java Code.

Wo ist das Problem? Wie können wir das besser machen? Ideen?

# Automatisierter Szenenaufbau

## Philosophisches

Eigentlich wollen wir gar nicht rechnen:

- Der *Computer* soll für uns rechnen.
- Wir wollen nur relativ angeben *wo das Objekt hin* soll.
  Z.B.: *2cm weiter nach rechts und um 180 Grad gedreht.*

$\rightarrow$ Trick: Wir verwenden immer das gleiche Dreieck und bugsieren es erst danach an Ort und Stelle.

Beispiel: Rotation eines Punktes entlang der x-Achse:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \\ 0 & \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

Weitere Rotationsmatrizen:

Rotation um Y:

$$R_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

Rotation um Z:

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

# Automatisierter Szenenaufbau

## 4x4 Matrizen

Obwohl wir uns in einem 3D-System bewegen, werden für Computergrafik häufig 4x4 Matrizen verwendet:

$$T_v = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Das ermöglicht Translation mithilfe von *Matrixmultiplikationen* anstelle von Vektoraddition:
$T_v p = p + v$ [1]

[1] Der Punkt wird dabei um die *homogene* Komponente, eine *1* an vierter Stelle, erweitert.

# Automatisierter Szenenaufbau

## 4x4 Matrizen

Dementsprechend verwenden wir 4x4 Matrizen zum Rotieren:

Translation um $v$:

$$\begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation um X:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation um Y:

$$\begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation um Z:

$$\begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Automatisierter Szenenaufbau

## Matrizen im Code

Initialisieren einer Rotationsmatrix: (Rotationsangabe in Grad)

```
Matrix.setRotateM(rotMat, 0, δ, vx, vy, vz);
```

Translation einer Matrix:

```
Matrix.translateM(translationMatrix, 0, .5f, .5f,
0);
```

Matrixmultiplikation, A x B

```
Matrix.multiplyMM(zielMat, 0, A, 0, B, 0);
```

**Aufgabe 3**:

- Verwenden Sie eine Rotations-Matrix um das ursprüngliche Dreieck der vorherigen Aufgabe kopfüber zu stellen.
- Generieren sie ein weiteres Dreieck, links zum Ursprung versetzt. Verwenden sie dazu eine Translations-Matrix.
- Verwenden Sie eine Rotations-Matrix um dieses neue Dreieck um 90 Grad um seinen Schwerpunkt zu drehen.

- *Bonusaufgabe*: Generieren per Schleife 12 kleinere Dreiecke die als Ziffernblatt kreisförmig angeordnet sind.

# Automatisierter Szenenaufbau

## Nachtrag

**Vorsicht mit der Reihenfolge von Matrixoperationen**:

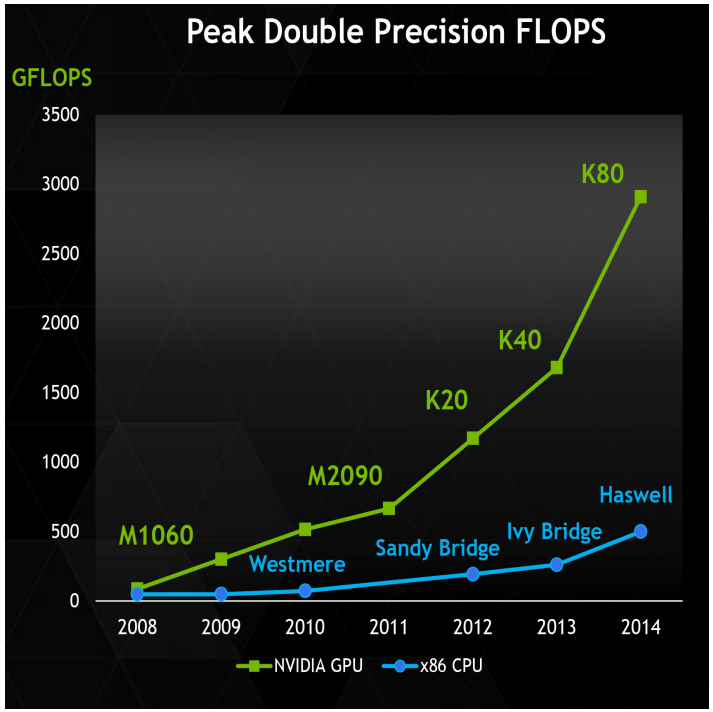
```
Matrix.rotateM(cameraPosMatrix, 0, 45, 0, 0, 1);
Matrix.translateM(cameraPosMatrix, 0, .5f, 0, 0);
```

$\neq$

```
Matrix.translateM(cameraPosMatrix, 0, .5f, 0, 0);
Matrix.rotateM(cameraPosMatrix, 0, 45, 0, 0, 1);
```

Was ist der Unterschied zwischen CPU und GPU?

- **C**entral **P**rocessing **U**nit
  - wenige Kerne
  - hohe Taktung
  - Kerne arbeiten *unabhängig*
- **G**raphics **P**rocessing **U**nit
  - viele Kerne
  - geringere Taktung
  - Kerne arbeiten lediglich *datenparallel*




http://www.anandtech.com/show/8729/nvidia-launches-tesla-k80-gk210-gpu

Zugriff auf Array per Thread-ID:

```
if(threadIdx.x > 0 && threadIdx.x < 4
    && threadIdx.y > 0 && threadIdx.y < 4)  {
    float newVal = 0;
    newVal += image[threadIdx.x-1][threadIdx.y-1];
    newVal += 2*image[threadIdx.x][threadIdx.y-1];
    newVal += image[threadIdx.x+1][threadIdx.y-1];
    newVal += 2*image[threadIdx.x-1][threadIdx.y];
    newVal += 4*image[threadIdx.x][threadIdx.y];
    newVal += 2*image[threadIdx.x+1][threadIdx.y];
    newVal += image[threadIdx.x-1][threadIdx.y+1];
    newVal += 2*image[threadIdx.x][threadIdx.y+1];
    newVal += image[threadIdx.x+1][threadIdx.y+1];

    outputImage[x*mxWidth+y] = newVal / 16; }
```

Gauß Filter:

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

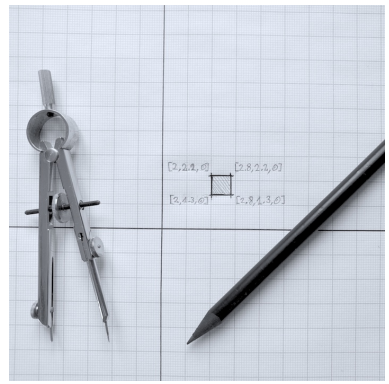
# Programm

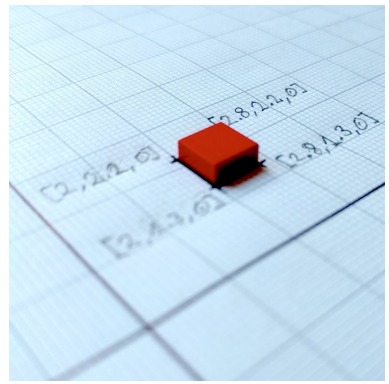
# Animierte Bewegungen

## Kurze Erinnerung...

Unsere mini-Pipeline hat 3 Schritte:

- Objekte platzieren ← *Hierum geht's*
- Betrachtungsort und -Winkel wählen
- Foto machen

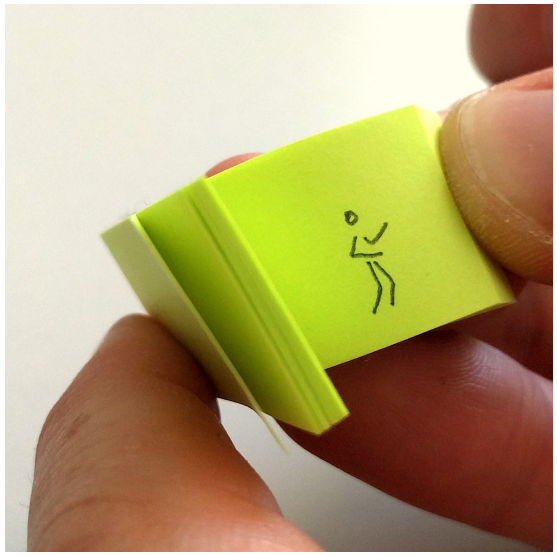




# Animierte Bewegungen

## Animationen

Eine Animation ist wie ein Daumenkino:



- Es gibt eine Sequenz von Einzelbildern
- Ein Bild beschreibt einen temporären Zustand
- Ein Bild erscheint nur sehr kurz

Es entsteht die *Illusion* einer flüssigen Bewegung.

# Animierte Bewegungen

## Animationen

Eine Animation ist **kein** Daumenkino:

- Wir wissen nicht wann das nächste Bild dran ist.
- Wir haben keine vorbereiteten Bilder.

**Problem**: Wir wissen nicht wann der Renderer anspringt.

**Ideen... ?**

# Animierte Bewegungen

## Animationen

**Problem**: Wir wissen nicht wann der Renderer anspringt.

- Wir fragen zur Laufzeit die Systemzeit ab.
- Wir bestimmen den relativen Fortschritt der Animation.
- Wir bauen eine Szene entsprechend des Fortschritts.

# Animierte Bewegungen

## Beispiel

Ein Dreieck soll einmal pro Sekunde rotieren.
Pseudocode im Renderer:

```
long fortschritt = zeitAnimationsStart - zeitJetzt;
float winkel = fortschritt / animationsdauer * 360;
...
Matrix.rotateM(cameraPosMatrix, 0, winkel, 0, 0, 1);
```

# Animierte Bewegungen

## Formeln

Je nach Formel kann die Systemzeit für fließende oder für sprunghafte Bewegungen verwendet werden:

- Was könnte das hier sein?
  `(int)(System.currentTimeMillis()/1000)%(60))*6`
- Oder das hier?
  `(float)(System.currentTimeMillis()%60000*0.006)`

# Animierte Bewegungen

## Exkurs: Renderer-Modi

Was ist der Unterschied zwischen *When Dirty* und *Continuously*?[2]

- **When Dirty**
  - Haben wir bislang verwendet
  - Szene wird nur neu gerendert, wenn wir manuell `requestRender()` aufrufen

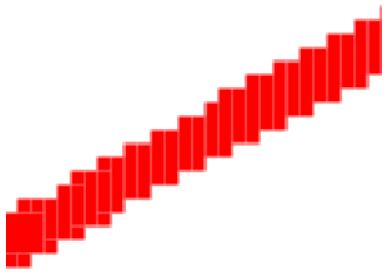
- **Continuously**
  - Brauchen wir für Animationen
  - Nächstes Bild wird erzeugt sobald aktuelles fertig

[2] Eintrag in `*SurfaceView`:
`setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);`

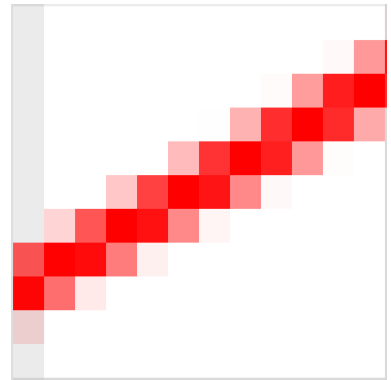
**Aufgabe 4**:

- Setzen sie den Renderer-Modus auf *Continuously*.
- Wandeln Sie ihre Szene zu einer Uhr (nur Sekundenzeiger) um.
- Testen Sie einen springenden und einen fließenden Zeiger.
- Fügen Sie Minuten- und Stundenzeiger hinzu.

**A**nti-**A**liasing ist eine Technik um Pixel-Artefakte zu minimieren.



**Kein AA**: Dichte Pixel, binäre Farbwahl.



**Aktives AA**: Grobe Pixel, proportionale Farbwahl.

# Exkurs

## Anti Aliasing

OpenGL stellt Anti-Aliasing bereit:

- Sie können es per Konfiguration aktivieren.
- Öffnen Sie die `SurfaceView` Klasse.
- Fügen Sie im Konstruktor den Aufruf zur Verwendung der Anti-Aliasing Konfiguration hinzu:
  ```
  setEGLConfigChooser(new AntiAliasedConfig());
  ```

# Programm

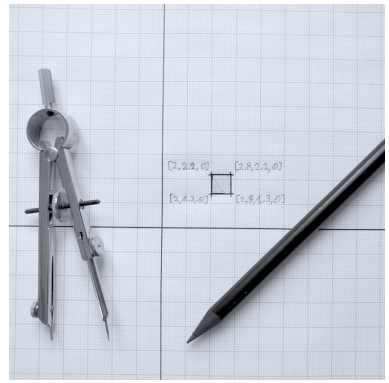
1. Minimale Pipeline
2. Automatisierter Szenenaufbau
3. Animierte Bewegungen
4. Goodies

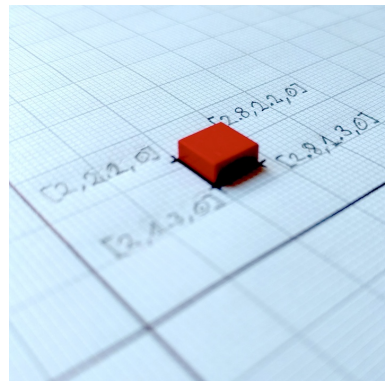
# Goodies

## Kurze Erinnerung...

Unsere mini-Pipeline hat 3 Schritte:

- Objekte platzieren
- Betrachtungsort und -Winkel wählen
  ↑ *Hierum geht's*
- Foto machen





# Goodies

## Sensoren

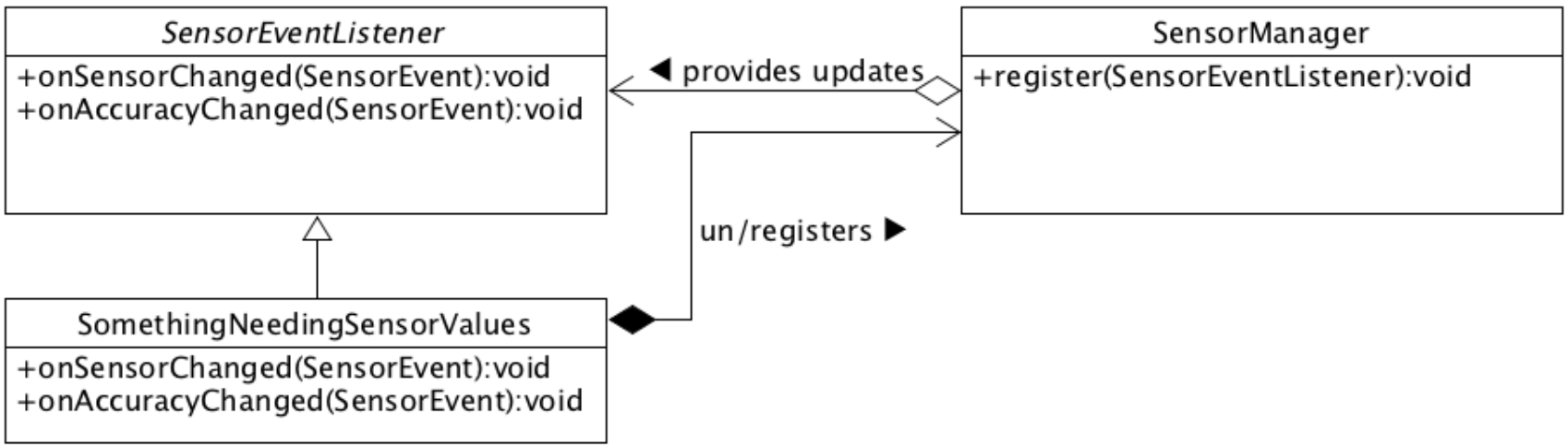
Physische Interaktion:

- Bis jetzt können wir nicht mit der App interagieren.
- Wir möchten die Ausgabe durch physische Interaktionen beeinflussen.
- **Normalerweise** erfolgt der Zugriff auf Sensorwerte entsprechend des Hollywood-Prinzips

# Goodies

## Exkurs: Observer Pattern

*Don't call us, we'll call you!*




Zwei Vorteile von Verzicht auf permanentes *Pollen*:

- Keine redundanten Anfragen, Ressourcen-freundlich.
- Geringere Latenzen, da Änderungen sofort propagiert werden.

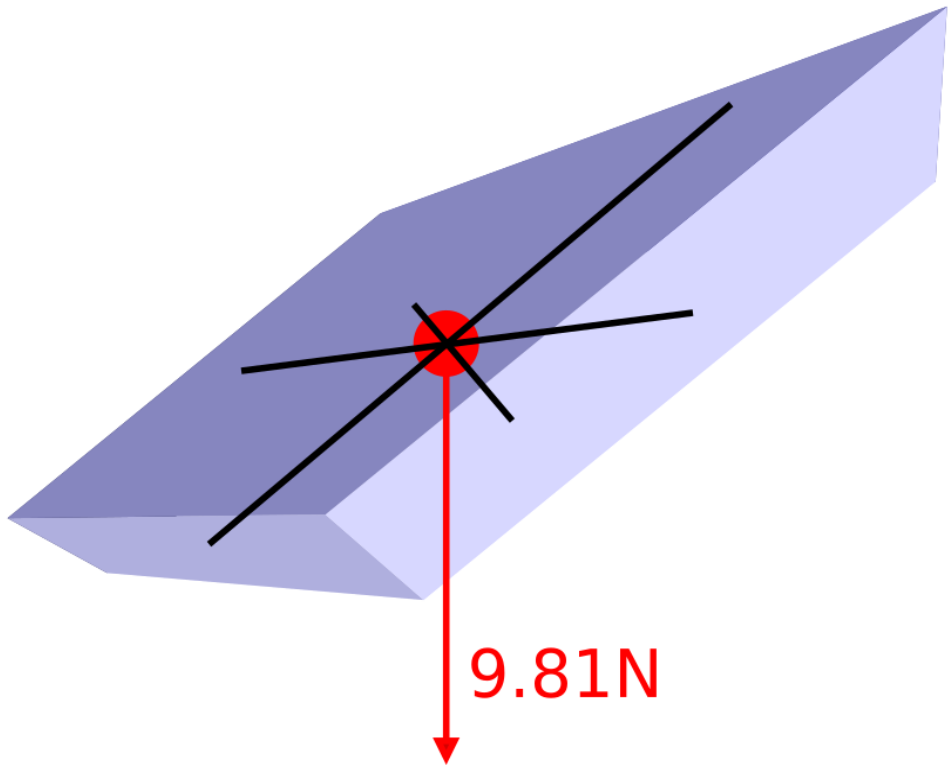
# Goodies

## Hollywood vs Framerate

Wegen des Update-Zyklus der Animationen (*Framerate*) können wir dieses Pattern nicht unangepasst anwenden:

- Wir implementieren den Listener daher als Puffer...
- ... und pollen vom Puffer, bei jeder UI-Iteration

Wir möchten mithilfe der Gravitations-Sensoren eine Bildstabilisierung implementieren:



**Aufgabe 4**:

- Übergeben sie von der Activity eine Instanz des Schwerkraft-Puffers an ihren Renderer. Testen sie diesen indem Sie bei jeder Renderer Iteration dessen Werte ausgeben.
- Kompensieren Sie den Haltewinkel des Geräts indem Sie die Kamera verlagern. Bewegen Sie dazu die Kamera planar über der Szene.
- Bonus: Ändern Sie die Bildstabilisierung dahingehend ab, dass der Haltewinkel durch eine Positionierung der Kamera auf einer Sphäre anstatt in der Ebene erfolgt.

# Danke

## Fragen, Wünsche, Anregungen?