



2D Spiele in Java

Teil 3



© 2012-18 by Thomas Helml



Inhalt

- IT Full Screen Mode vs. Window Mode
- IT Actives vs. Passives Rendering
- IT 2D- Hardwarebeschleunigung
- IT Steuerung
- IT Game-Loop



Fullscreen vs Windowmode

- ① Spiele laufen üblicherweise Fullscreen
- ① Hardwarebeschleunigung teilw. nur im Fullscreenmode
- ① „Zeichnen“ wird vom Programmierer übernommen



Active vs. Passive Rendering

- ① in AWT wird das Zeichnen des Fensters vom OS ausgelöst
 - ① Programm weiß nicht, wann der Benutzer das Fenster vergrößert, verschiebt, ...

```
public void paint(Graphics g) {  
    // Use g to draw my Component  
}
```

- ① sogenanntes „Passives Rendering“



Actives vs. Passives Rendering

- ① bei der Spieleentwicklung wird das Rendern aktiv betrieben
- ① d.h. wir bestimmen selbst
 - ① wann etwas gezeichnet wird
 - ① was gezeichnet wird
- ① Automatismus wird „deaktiviert“



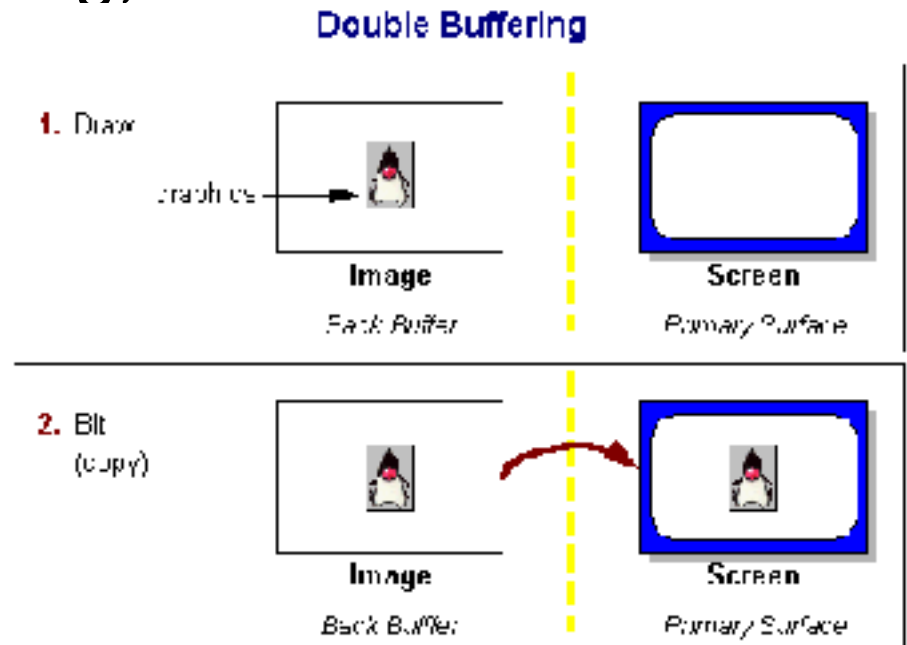
Hardwarebeschleunigung

- ① Bewegung am Bildschirm
 - ① neu Zeichnen des gesamten Bildes mit geänderter Szene
 - ① Vorher: altes Bild löschen
 - ① Problem:
 - ① im schlimmsten Fall: Zeichnen kann beobachtet werden
 - ① Löschen führt zu Flackern!

IT Double Buffering

- IT das Zeichnen des Bildes erfolgt vollständig
- IT erst wenn vollständig, wird das Bild in den GrafikRAM kopiert

IT Blit = kopieren





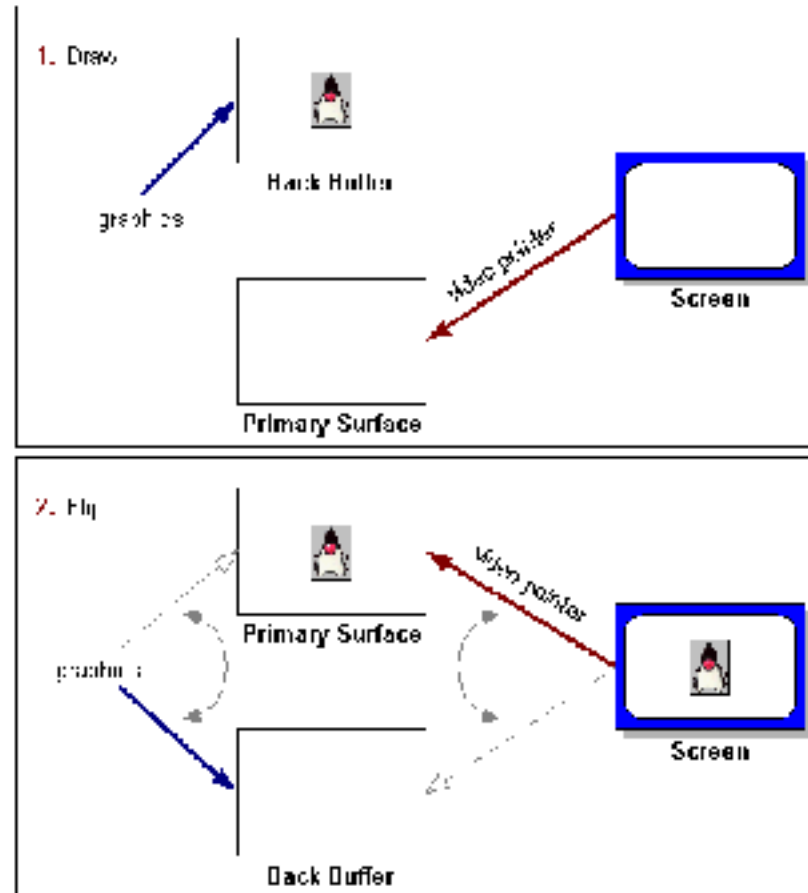
Hardwarebeschleunigung

IT Page Flipping

- IT manche GraKa haben einen sog. „Video Pointer“
- IT dieser zeigt auf eine Adresse im VideoRAM der angezeigt wird
- IT Video Pointer kann einfach auf eine neue Adresse gesetzt werden
- IT es können mehrere Bilder im Vorhinein berechnet werden

Hardwarebeschleunigung

Page Flipping





Hardwarebeschleunigung

- ① Rechenbeispiel:
 - ① 60 fps
 - ① Auflösung 1280x1024, 32Bit Farbtiefe
 - ① ca. 5MB pro Frame
 - ① $\Rightarrow 60 \times 5\text{MB} = 300\text{MB}$ Datentransfer am Bus
(RAM \rightarrow VRAM)



Hardwarebeschleunigung

- ⑩ ab Java 1.5: VolatileImage
- ⑩ Bilder werden direkt im VideoRAM geladen
 - ⑩ Vorteil: Bus wird entlastet!
 - ⑩ Nachteil: sie können „verloren“ gehen
 - ⑩ andere App geht Fullscreen
 - ⑩ Taskmanager wird geöffnet
 - ⑩ ...



Tastenabfrage

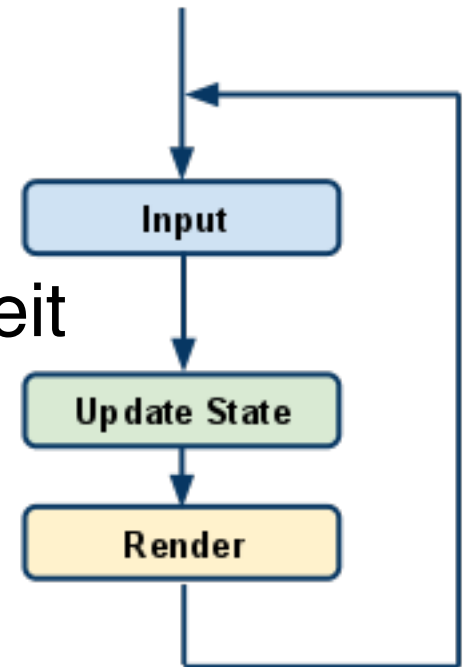
- ❶ für die Steuerung müssen die Key-Events abgefragt werden: Event bei
 - ❶ Taste wird gedrückt
 - ❷ Taste wird losgelassen
 - ❸ Taste wird „eingegeben“

```
public class GameKeyListener extends KeyAdapter
{
    public void keyPressed(KeyEvent e)
    {
        if (e.getKeyCode() == KeyEvent.VK_LEFT) left = true;
        if (e.getKeyCode() == KeyEvent.VK_RIGHT) right = true;
        if (e.getKeyCode() == KeyEvent.VK_SPACE) fire = true;
        if (e.getKeyChar() == 27) esc = true;
    }
}
```



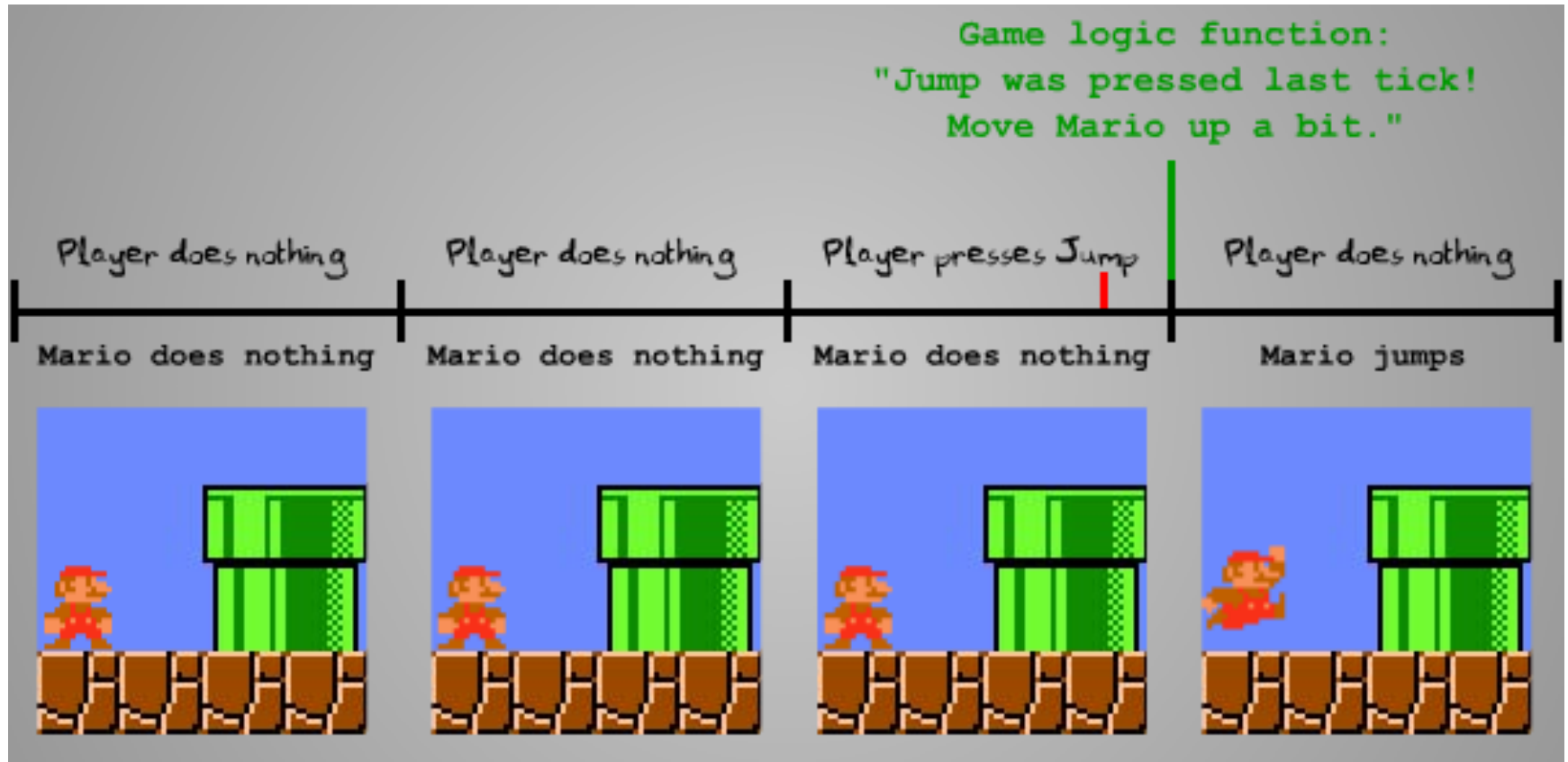
Game Loop

- IT die Game Loop ist zuständig für
 - IT Zeichnen der Szene
 - IT Abfragen der Tasten
 - IT Spiellogik
 - IT Kontrolle der Spielgeschwindigkeit





Game Loop





Game Loop

- ① Annahme: Action Spiel mit 60FPS
 - ① innerhalb von 60FPS muss
 - ① schnell auf Benutzeraktionen reagiert werden (Tastenabfragen auswerten)
 - ① Gravitation berechnen
 - ① KI berechnen
 - ① Grafik zeichnen
 - ① ...
 - ① Was ist, wenn Rechner zu schwach für diese Zeitvorgabe ist?



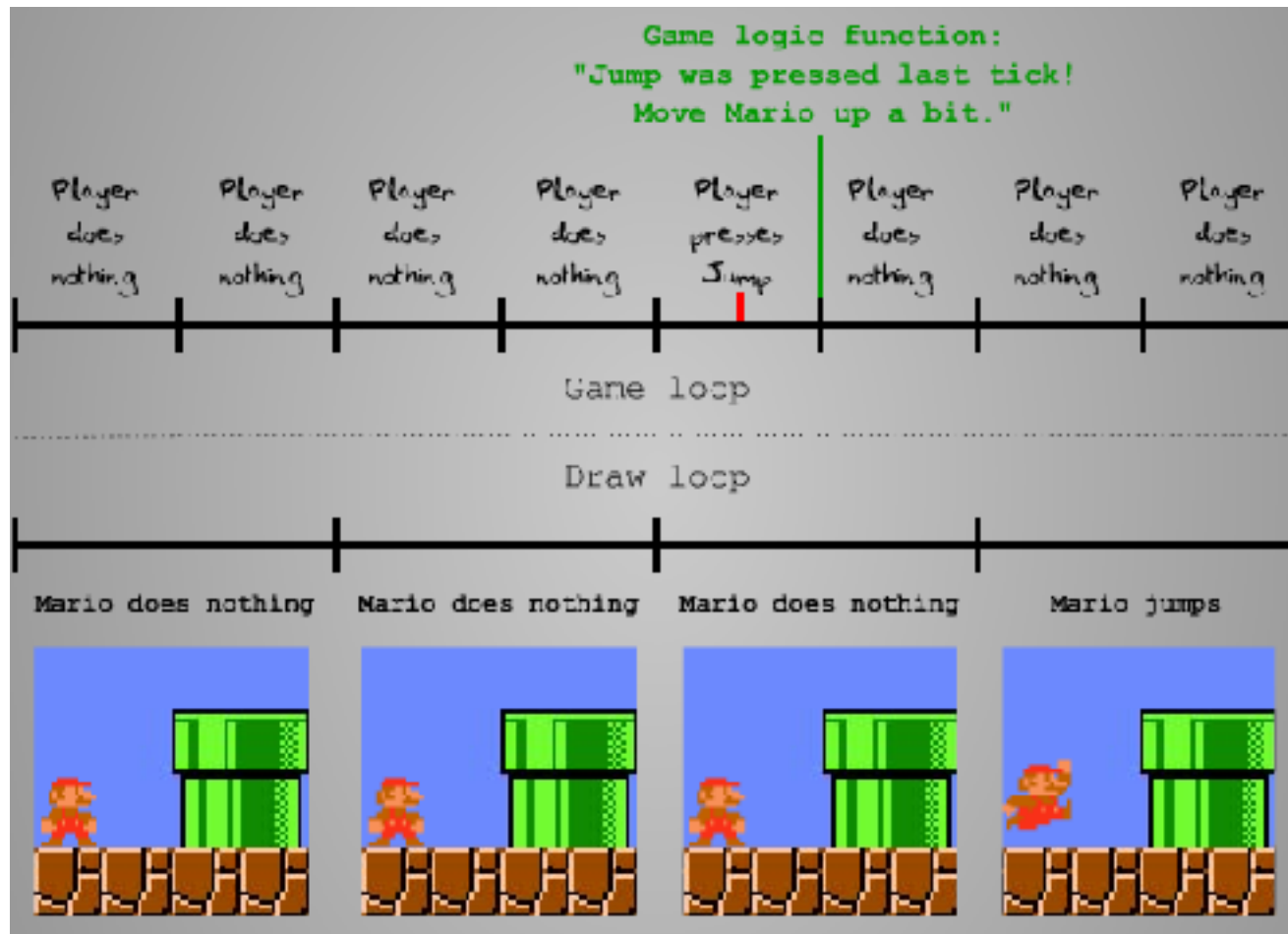
Game Loop

IT Lösung:

- IT FPS müssen weniger werden
- IT sprich: es wird nicht so oft gezeichnet
- IT ABER:
 - IT Spiellogik läuft weiterhin schneller
- IT -> Game Loop + Draw Loop!



Game Loop





Referenzen

IT Java Tutorials:

- IT <http://docs.oracle.com/javase/tutorial/extra/fullscreen/index.html>
- IT <http://www.petrastumpf.de/michael/Wissen/SpieleprogrammierungInJava.pdf>
- IT <http://www.javacodegeeks.com/2011/06/android-game-development-tutorials.html>

IT Volatile Image:

- IT <http://content.gpwiki.org/index.php/Java:Tutorials:VolatileImage>

IT Game Loop:

- IT <http://active.tutsplus.com/tutorials/games/understanding-the-game-loop-basix/>