LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
LEHR- UND FORSCHUNGSEINHEIT MEDIENINFORMATIK

Prof. Dr. Andreas Butz

Dipl.-Medieninf. Hendrik Richter
Dipl.-Medieninf. Raphael Wimmer

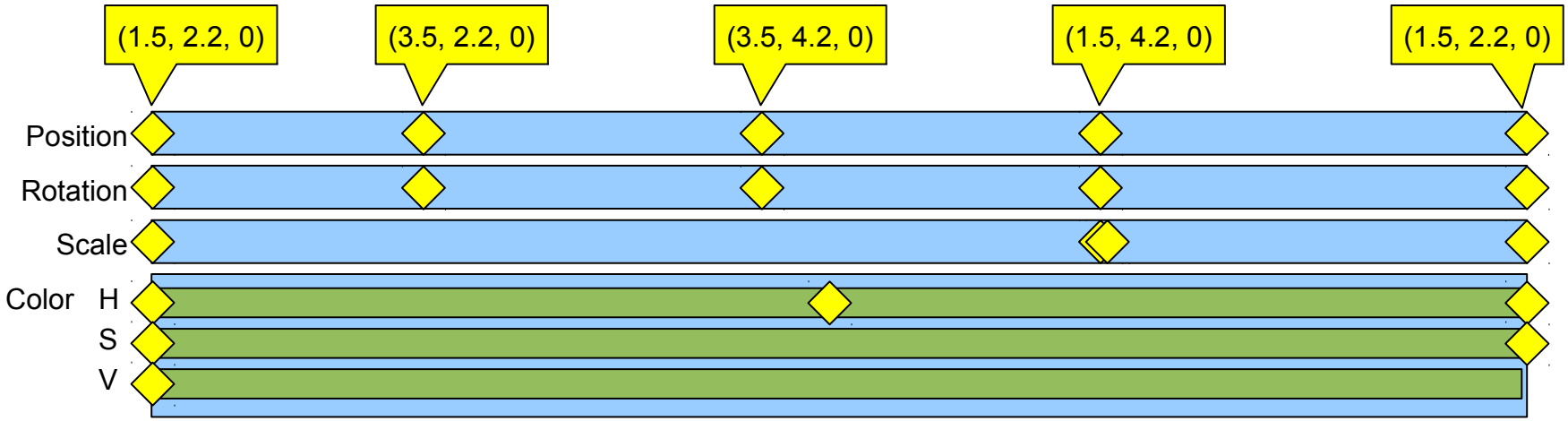
Computergrafik 1 Übung 8

Animation

http://www.michael-small.co.uk/images/Game1a.gif

Animation

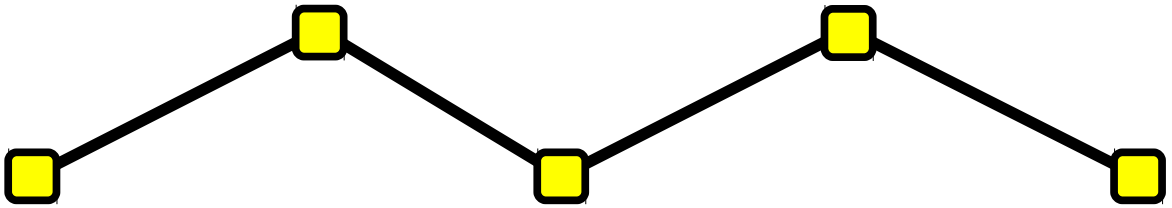
Keyframe-Animation



definiert Eigenschaften, nicht Transformationen

Interpolation zwischen zwei Keyframes

Pfadanimation



$$x = x_0 + \frac{t - t_0}{t_1 - t_0}(x_1 - x_0), y = y_0 + \frac{t - t_0}{t_1 - t_0}(y_1 - y_0)$$

Mesh-Animation

Morphen zwischen verschiedenen Mesh-Zuständen

1:1-Zuordnung der Vertices nötig

Bsp (HLSL):

```
float4 position =
   (1.0f - interp) * vertexIn.prevPositionKey
    + interp * vertexIn.nextPositionKey;
```

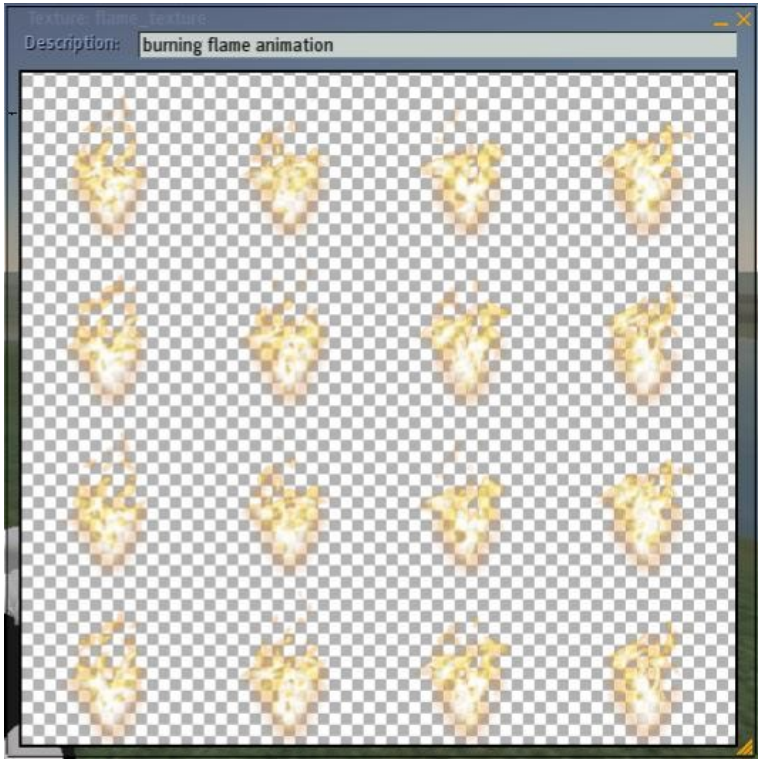
Morphing mit gewichteten Targets:

→ für alle Vertices Differenzvektoren zwischen Normalpose und Targets berechnen

→ Differenzvektoren gewichtet verrechnen und auf Normalpose anwenden

Bild + Code: http://http.developer.nvidia.com/GPUGems/gpugems_ch04.html

Textur-Animation



animierte Textur

+ wenig Rechenleistung notwendig
+ sehr realistisch
+ sehr flexibel

- kein perspektivischer 3D-Effekt
- keine Beleuchtungseffekte

z.B. verwendbar für Explosionen, Gesicht, Meer, Himmel

http://ce.bromley.ac.uk/lsl/kan-ed/copy/

Animation von Rotationen

Mit Winkeln:

→ Interpolation der einzelnen Winkel

Mit Rotationsmatrizen:

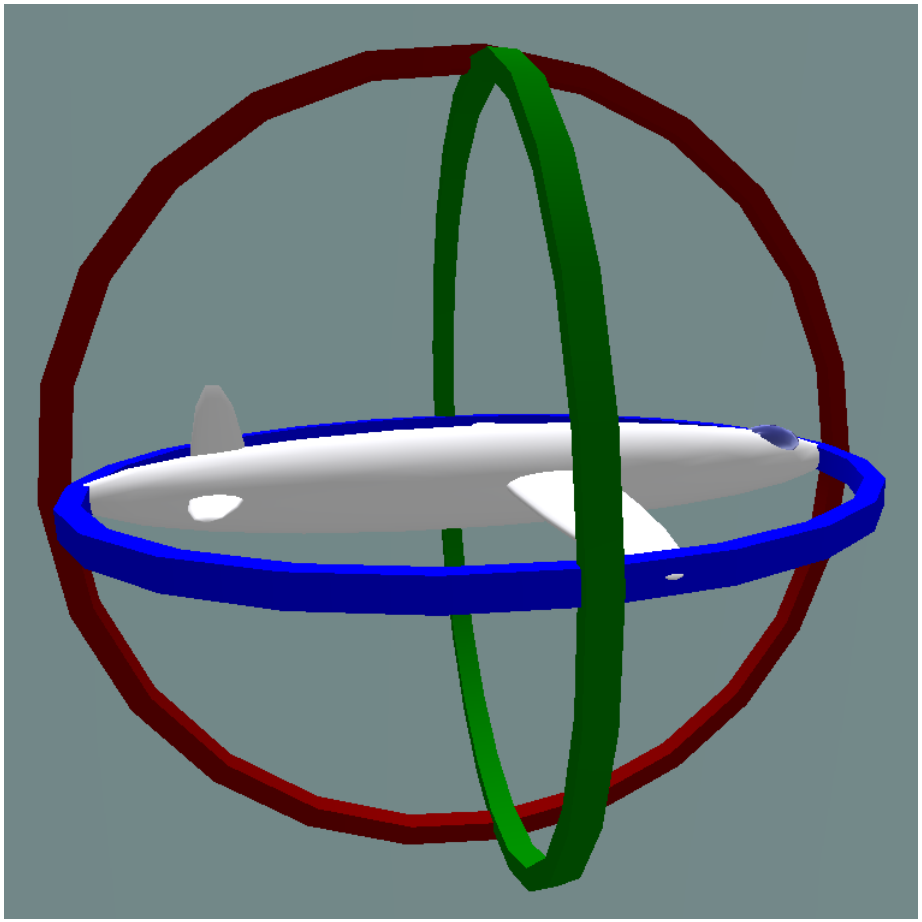
→ Interpolation der einzelnen Komponenten?

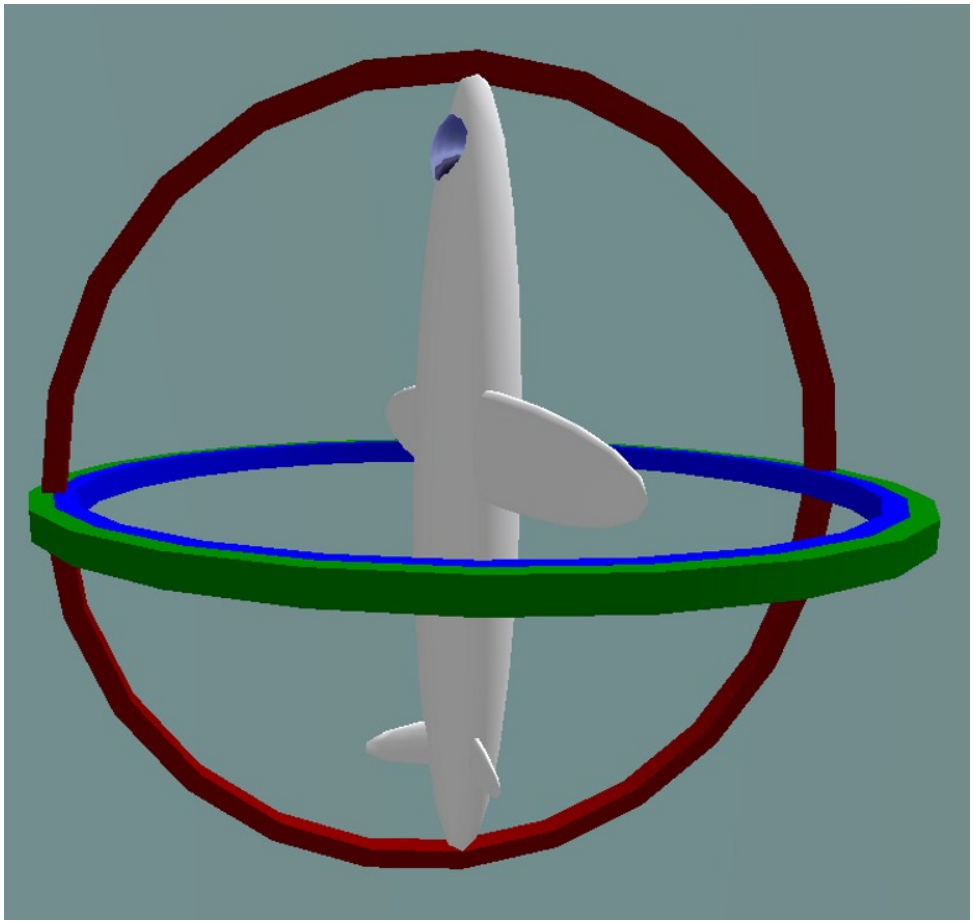
→ Extraktion der Rotationswinkel, Interpolation derselben, Erzeugen einer neuen Rotationsmatrix

→ umständlich.

Weiteres Problem: Gimbal Lock

Gimbal Lock





http://commons.wikimedia.org/wiki/File:No_gimbal_lock.png

http://commons.wikimedia.org/wiki/File:Gimbal_lock.png

Quaternionen to the Rescue

Komplexe Zahl: $w + v = (w + xi + yj + zk)$

mit $i^2 = j^2 = k^2 = ijk = -1$

Konjugiertes Quaternion: $q^* = w - xi - yj - zk$

Norm/Betrag: $|q| = \sqrt{w^2 + x^2 + y^2 + z^2}$, $|q|^2 = q \cdot q^*$

Einheitsquaternion $\rightarrow$ $|q| = 1$

Inverse: $q^{-1} = q^* / |q|$

Produkt $q_1 q_2 = [w_1 w_2 - v_1 \cdot v_2, \quad v_1 \times v_2 + w1 \cdot v2 + w2 \cdot v1]$

$\rightarrow$ assoziativ, distributiv, aber nicht kommutativ

Polarform: $q = r \cdot (\cos \varphi + i \cdot \sin \varphi + j \cdot \sin \varphi + k \cdot \sin \varphi)$

alle folgenden Formeln aus: Thomas Koch: Rotationen mit Quaternionen in der Computergrafik

Rotation mit Quaternionen

Vektor v → p = [0, v_x, v_y, v_z]

q = r·(cos φ + i·sin φ + j·sin φ + k·sin φ)

→ q(45°, Y-Achse) = (cos(45°), (0, 1, 0) * sin(45°))
= (cos(45°), 0, sin(45°), 0))

Rotation: p' = q * p * q^{-1}

Rotation um 2*φ (q und q^{-1})!

Quaternionen & Rotationsmatrizen

Quaternion zu Rotationsmatrix:

$$R=\begin{pmatrix} 1-2\cdot(y^2+z^2) & 2\cdot(x\cdot y-w\cdot z) & 2\cdot(x\cdot z+w\cdot y) & 0 \\ 2\cdot(x\cdot y+w\cdot z) & 1-2\cdot(x^2+z^2) & 2\cdot(y\cdot z-w\cdot x) & 0 \\ 2\cdot(x\cdot z-w\cdot y) & 2\cdot(y\cdot z+w\cdot x) & 1-2\cdot(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Slerp: Spherical Linear Interpolation

$$\text{Slerp}\left(q_1, q_2, t\right) = \frac{q_1 \cdot \sin\left(\left(1-t\right)\Omega\right) + q_2 \cdot \sin\left(t \cdot \Omega\right)}{\sin \Omega},$$

$$\cos\left(\Omega\right) = q_1 \cdot q_2$$

Interpolationsschritt: t = [0,1]

Quaternionen vs. Matrizen

+ weniger Speicherbedarf
+ etwas weniger Rechenoperationen nötig
+ kein Gimbal Lock
+ elegante Interpolation/Animation der Rotation

- kein OpenGL-Support

Vielen Dank!