

Perspektivisch korrigierte Box-Simulation mittels Microsoft Kinect

Seminarbeitrag Scientific Visualisation

Beneke, Razzaq, Unger

Fachbereich Mathematik und Technik
RheinAhrCampus Remagen

9. Juli 2012



- 1 Motivation**
- 2 Kinect und ihre Funktionsweise**
- 3 3D-Effekt durch perspektivische Transformation**
- 4 Physikalisches Modell**
- 5 Vorführung**
- 6 Resümee**

Zielsetzung

- Bewegungsdetektion mittels Microsoft Kinect
- Modellierung einer dreidimensionalen Umgebung mittels OpenGL
- Perspektivische Transformation des 3D-Raums abhängig von der Kopfposition des Probanden → 3D-Effekt
- Kopplung eines physikalischen Modells mit den Bewegungsabläufen des Probanden → Boxsack-Simulation

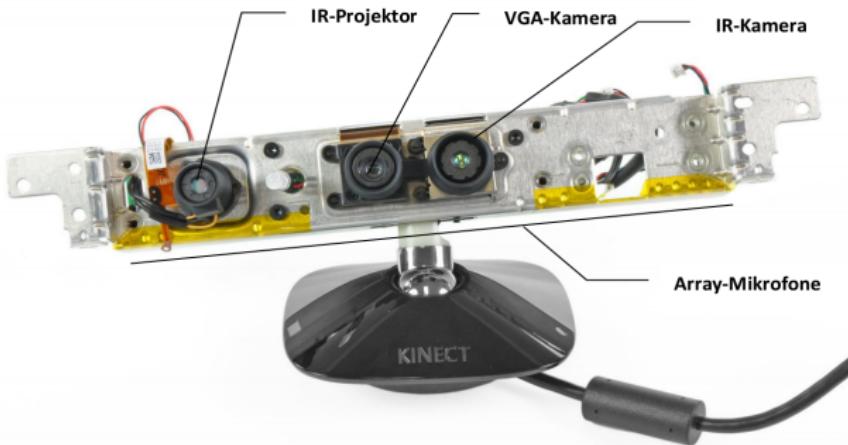
Allgemeines



Quelle: www.amazon.de

- erschien im November 2010
- Peripheriegerät zur XBox 360 entwickelt von PrimeSense
- Erkennung gespeicherter Strukturen möglich
- ermöglicht Eingaben über Gesten und Bewegungen

Komponenten der Kinect



Quelle: wiki.zimt.uni-siegen.de/fertigungautomatisierung/images/5/54/S910310_abb2.png

Die Kinect besteht aus folgenden Komponenten

- Infrarotlaser (780nm, Laserklasse 1)
- Infrarotsensor
- RGB-Kamera (640x512 Pixel, 30 fps, 32-bit Farbtiefe)
- Mikrofon-Array

Prinzip zur Bilderstellung

Prinzip eines Structured-Light-3D-Scanner

Kinect verwendet spezielle Art: Das sogenannte Light-Coding

- Laser erzeugt mit infrarotem strukturiertem Licht (IRSL) ein Muster mit unterschiedlicher Punktintensität
- verwendet dabei ein Set von Beugungsgittern vor der Lichtquelle

Light Coding



Quelle: <http://www.scribd.com/doc/56470872/3/Light-Coding-with-the-Kinect>

Light Coding

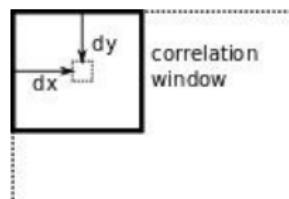


Quelle: http://rrt.fh-wels.at/publications/technical_papers/MP1_Kinect_Kofler_2011.pdf

- Wiederholendes Muster (3x3) mit 211x165 Pixeln
- Gesamte Pixelzahl 633x495

Tiefenbestimmung

Kinect gewinnt Daten über Triangulation des Grundmusters mit dem durch den Infrarotsensor aufgenommenem Bild



Quelle: http://rrt.fh-wels.at/publications/technical_papers/MP1_Kinect_Kofler_2011.pdf

Vergleich eines Pixels mit seinen 64 nächsten Nachbarn

Softwareansteuerung der Kinect

- libfreenect ermöglicht Ansteuerung der Kinect über einen PC
- Microsoft gibt offiziellen Support mit Kinect SDK

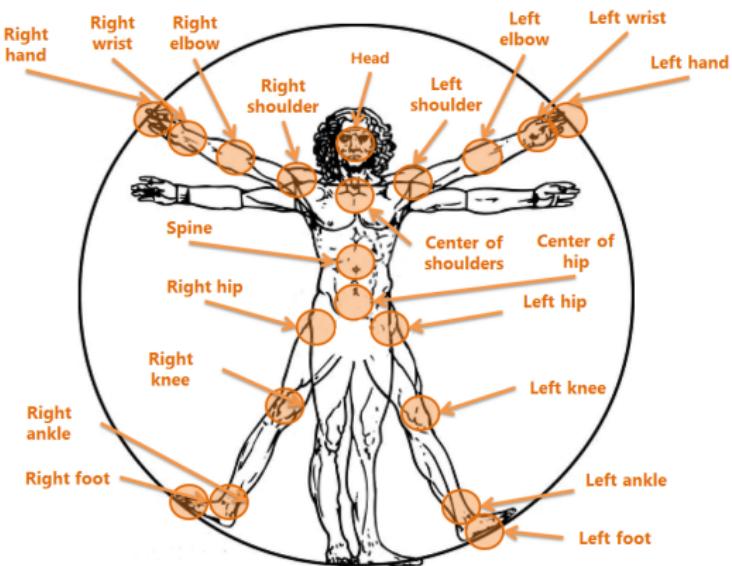
Im Rahmen des Projektes verwendeten wir:

- 1 Python
- 2 pyKinect (basiert auf pyGame)

Skelett-Tracking

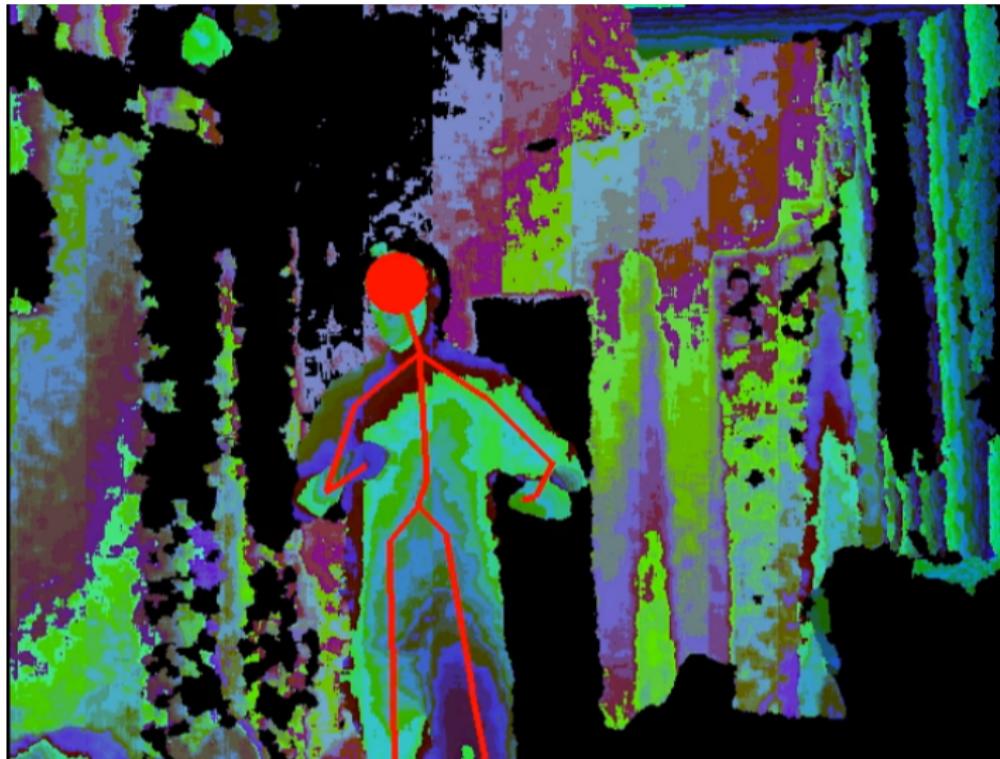
Kinect SDK liefert die Möglichkeit Körperteile zu erkennen und Positiondaten zuzuordnen

Ausgabe eines Grundskelettes leicht möglich.



Quelle: <http://praveenitech.files.wordpress.com>

Skelett-Tracking



Matrixtransformationen in OpenGL

Bsp.: Translation um x , y und z auf zwei Wegen.

1) Bordmittel

```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
glTranslatef( x, y, z)  
renderObject()
```

2) Matrixtransformation

```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
T = array([[ 1, 0, 0, x]  
           [ 0, 1, 0, y]  
           [ 0, 0, 1, z]  
           [ 0, 0, 0, 1]])  
glMultMatrix( T)  
renderObject()
```

Beide Wege führen identische Matrixtransformation aus.

Matrixtransformationen in OpenGL

Bsp.: Translation um x , y und z auf zwei Wegen.

1) Bordmittel

```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
glTranslatef( x, y, z)  
renderObject()
```

2) Matrixtransformation

```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
T = array([[ 1, 0, 0, x]  
          [ 0, 1, 0, y]  
          [ 0, 0, 1, z]  
          [ 0, 0, 0, 1]])  
glMultMatrix( T)  
renderObject()
```

Beide Wege führen identische Matrixtransformation aus.

Matrixtransformationen in OpenGL

Bsp.: Translation um x , y und z auf zwei Wegen.

1) Bordmittel

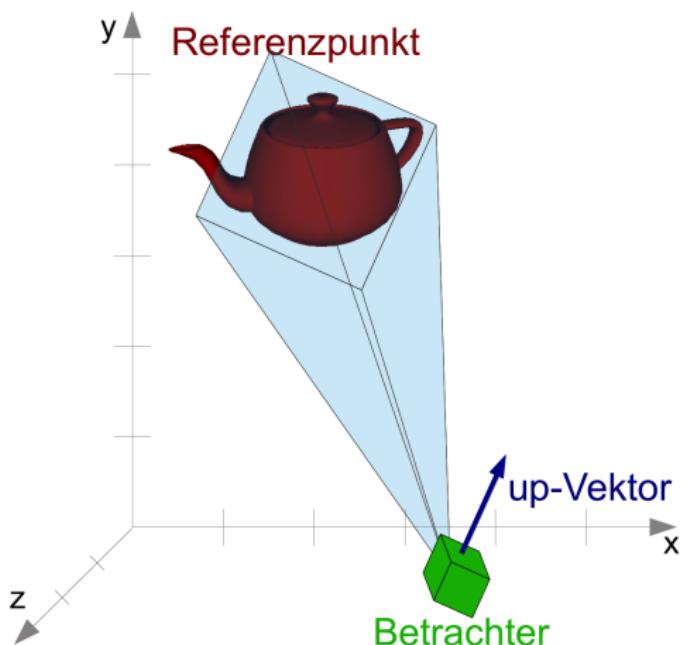
```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
glTranslatef( x, y, z)  
renderObject()
```

2) Matrixtransformation

```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
T = array([[ 1, 0, 0, x]  
          [ 0, 1, 0, y]  
          [ 0, 0, 1, z]  
          [ 0, 0, 0, 1]])  
glMultMatrix( T)  
renderObject()
```

Beide Wege führen identische Matrixtransformation aus.

Drei entscheidende Vektoren



Betrachter

$$\vec{eye} = (eye_x, eye_y, eye_z)^T$$

Referenzpunkt

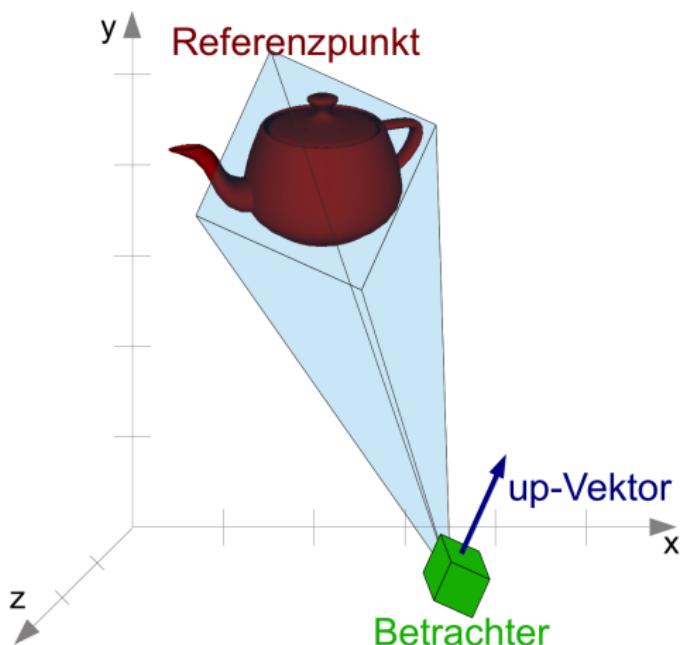
$$\vec{ref} = (ref_x, ref_y, ref_z)^T$$

Definition "oben"

$$\vec{up} = (up_x, up_y, up_z)^T$$

Quelle: <http://wiki.delphigl.com/index.php/gluLookAt>

Drei entscheidende Vektoren



Betrachter

$$\vec{eye} = (eye_x, eye_y, eye_z)^T$$

Referenzpunkt

$$\vec{ref} = (ref_x, ref_y, ref_z)^T$$

Definition "oben"

$$\vec{up} = (up_x, up_y, up_z)^T$$

Quelle: <http://wiki.delphigl.com/index.php/gluLookAt>

Herleitung der Transformationsmatrix

$$\begin{aligned}\vec{f} &= r\vec{e}_f - e\vec{y}e \\ \vec{s} &= \vec{f}' \times \vec{u}\vec{p}' \\ \vec{u} &= \vec{s} \times \vec{f}\end{aligned}$$

Betrachterfokus/ "Tiefe"
"Breite"
"Höhe"

Transformationsmatrix

$$M = \begin{pmatrix} \vec{s}_x & \vec{s}_y & \vec{s}_z & 0 \\ \vec{u}_x & \vec{u}_y & \vec{u}_z & 0 \\ -\vec{f}_x & -\vec{f}_y & -\vec{f}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Spalten: (x-Achse y-Achse z-Achse Translation)

$\vec{f}' = \vec{f}/|\vec{f}|$ ist der normierte Vektor von f

Quelle: <https://www.opengl.org/sdk/docs/man/xhtml/gluLookAt.xml>

Herleitung der Transformationsmatrix

$$\begin{aligned}\vec{f} &= r\vec{e}_f - e\vec{y}e && \text{Betrachterfokus/ "Tiefe"} \\ \vec{s} &= \vec{f}' \times \vec{u}\vec{p}' && \text{"Breite"} \\ \vec{u} &= \vec{s} \times \vec{f} && \text{"Höhe"}\end{aligned}$$

Transformationsmatrix

$$M = \begin{pmatrix} \vec{s}_x & \vec{s}_y & \vec{s}_z & 0 \\ \vec{u}_x & \vec{u}_y & \vec{u}_z & 0 \\ -\vec{f}_x & -\vec{f}_y & -\vec{f}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Spalten: (x-Achse y-Achse z-Achse Translation)

$\vec{f}' = \vec{f}/|\vec{f}|$ ist der normierte Vektor von f

Quelle: <https://www.opengl.org/sdk/docs/man/xhtml/gluLookAt.xml>

Herleitung der Transformationsmatrix

$$\begin{aligned}\vec{f} &= r\vec{e}_f - e\vec{y}e && \text{Betrachterfokus/ "Tiefe"} \\ \vec{s} &= \vec{f}' \times \vec{u}\vec{p}' && \text{"Breite"} \\ \vec{u} &= \vec{s} \times \vec{f} && \text{"Höhe"}\end{aligned}$$

Transformationsmatrix

$$M = \begin{pmatrix} \vec{s}_x & \vec{s}_y & \vec{s}_z & 0 \\ \vec{u}_x & \vec{u}_y & \vec{u}_z & 0 \\ -\vec{f}_x & -\vec{f}_y & -\vec{f}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Spalten: (x-Achse y-Achse z-Achse Translation)

$\vec{f}' = \vec{f}/|\vec{f}|$ ist der normierte Vektor von f

Quelle: <https://www.opengl.org/sdk/docs/man/xhtml/gluLookAt.xml>

Anwenden der Matrix

Matrixmultiplikation

```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
  
# MATRIX HIER GENERIEREN  
  
glMulMatrixf( M)  
glTranslatef( -eye_x, -eye_y, -eye_z)  
renderObject()
```

oder einfacher ...

OpenGL Utility Library (GLU)

```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
gluLookAt( eye_x, eye_y, eye_z, ref_x, ref_y, ref_z, up_x,  
           up_y, up_z)  
renderObject()
```

Anwenden der Matrix

Matrixmultiplikation

```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
  
# MATRIX HIER GENERIEREN  
  
glMulMatrixf( M)  
glTranslatef( -eye_x, -eye_y, -eye_z)  
renderObject()
```

oder einfacher ...

OpenGL Utility Library (GLU)

```
glMatrixMode( GL_MODELVIEW); glLoadIdentity()  
gluLookAt( eye_x, eye_y, eye_z, ref_x, ref_y, ref_z, up_x,  
           up_y, up_z)  
renderObject()
```

Kopplung mit Microsoft Kinect

```
# Betrachterposition von Kinect in x- und y- Achse
HeadPos = nui.SkeletonEngine.skeleton_to_depth_image(
            data.SkeletonPositions[ JointId.Head], 640, 480)
cam = [ ( HeadPos[0] / width), ( HeadPos[1] / height)]

# Perspektivische Transformation
gluLookAt( -cam[0], cam[1], cam[2],
            ref[0], ref[1], ref[2], 0, 1, 0)

# Szene rendern an Position center
renderScene( ref)
```

Kopplung mit Microsoft Kinect

```
# Betrachterposition von Kinect in x- und y- Achse
HeadPos = nui.SkeletonEngine.skeleton_to_depth_image(
            data.SkeletonPositions[ JointId.Head], 640, 480)
cam = [ ( HeadPos[0] / width), ( HeadPos[1] / height)]

# Perspektivische Transformation
gluLookAt( -cam[0], cam[1], cam[2],
            ref[0], ref[1], ref[2], 0, 1, 0)

# Szene rendern an Position center
renderScene( ref)
```

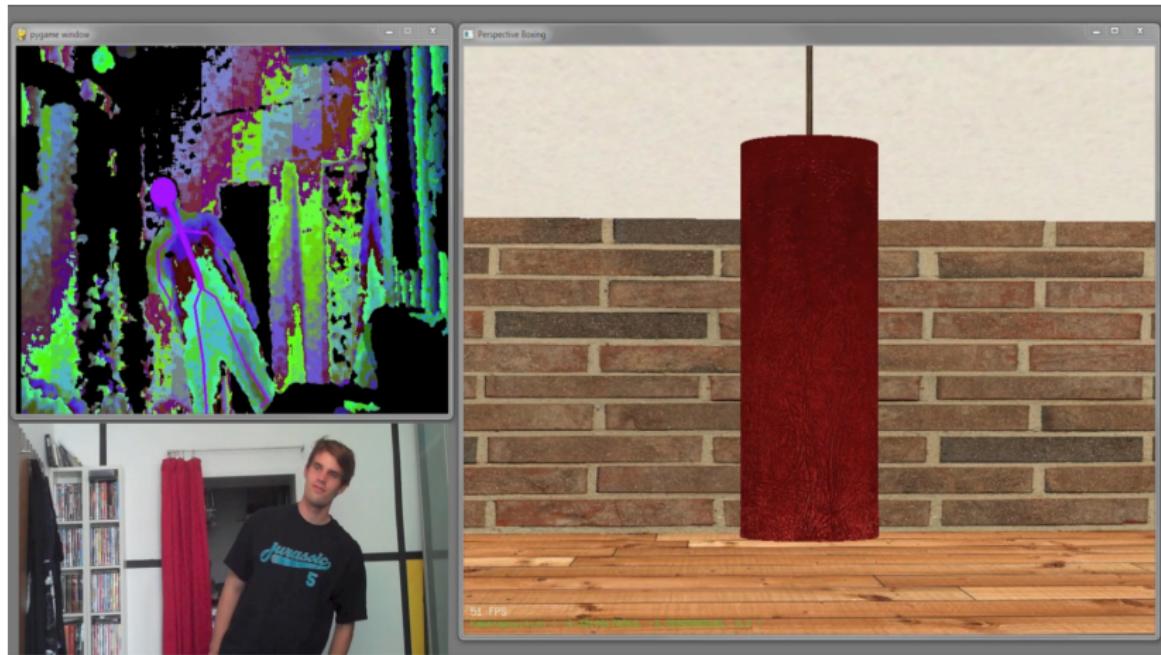
Kopplung mit Microsoft Kinect

```
# Betrachterposition von Kinect in x- und y- Achse
HeadPos = nui.SkeletonEngine.skeleton_to_depth_image(
            data.SkeletonPositions[ JointId.Head], 640, 480)
cam = [ ( HeadPos[0] / width), ( HeadPos[1] / height)]

# Perspektivische Transformation
gluLookAt( -cam[0], cam[1], cam[2],
            ref[0], ref[1], ref[2], 0, 1, 0)

# Szene rendern an Position center
renderScene( ref)
```

Videodemonstration



Mathematisches Pendel

$$\ddot{\varphi}(t) + \alpha\dot{\varphi}(t) + \beta \sin(\varphi(t)) = 0$$

$\alpha = \frac{g}{l}$ Auslenkung

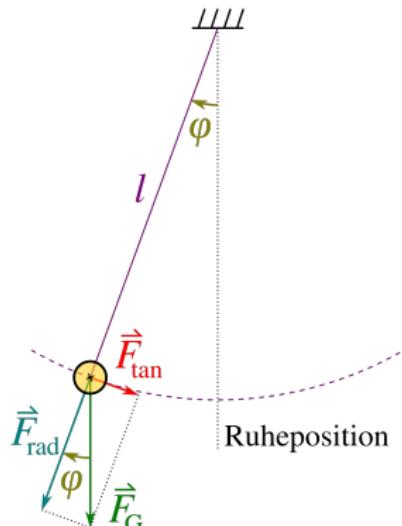
$\beta = \frac{\mu}{mL^2}$ Dämpfung

g = Erdbeschleunigung

m = Pendelmasse

l = Pendellänge

μ = viskose Reibung



Quelle: wikipedia.org

Lösung mit Euler-Verfahren

- Euler-Verfahren

$$\vec{x}^{k+1} = \vec{x}^k + h \vec{f}(\vec{x}^k, t^k)$$

- Zustandsvektor für mathematisches Pendel

$$\vec{x} = [\varphi(t), \dot{\varphi}(t)]^T$$

- Funktionsterm

$$\vec{f} = [\dot{\varphi}(t), -\beta \dot{\varphi}(t) - \alpha \sin(\varphi(t))]^T$$

- Schrittweite

$$h = dt$$

Schwingung bei horizontalem Schlag

```
# Normierte Koordinate
pos = handPos.x / width

# Geschwindigkeit
v = 0.0
if( lastPos - pos >= 0.05):
    v = pos / dt
lastPos = pos

# Winkelgeschwindigkeit
omega_neu = omega + v
omega += dt * (-alpha * math.sin(phi) - beta * omega_neu)

# Winkel
phi = phi + dt * omega_neu
phi = phi / math.pi * 180.

# Translation/ Rotation
glTranslatef( center.x, center.y, center.z)
glRotatef( phi, 0.0, 0.0, 1.0)
renderBag()
```

Schwingung bei horizontalem Schlag

```
# Normierte Koordinate
pos = handPos.x / width

# Geschwindigkeit
v = 0.0
if( lastPos - pos >= 0.05):
    v = pos / dt
lastPos = pos

# Winkelgeschwindigkeit
omega_neu = omega + v
omega += dt * (-alpha * math.sin(phi) - beta * omega_neu)

# Winkel
phi = phi + dt * omega_neu
phi = phi / math.pi * 180.

# Translation/ Rotation
glTranslatef( center.x, center.y, center.z)
glRotatef( phi, 0.0, 0.0, 1.0)
renderBag()
```

Schwingung bei horizontalem Schlag

```
# Normierte Koordinate
pos = handPos.x / width

# Geschwindigkeit
v = 0.0
if( lastPos - pos >= 0.05):
    v = pos / dt
lastPos = pos

# Winkelgeschwindigkeit
omega_neu = omega + v
omega += dt * (-alpha * math.sin(phi) - beta * omega_neu)

# Winkel
phi = phi + dt * omega_neu
phi = phi / math.pi * 180.

# Translation/ Rotation
glTranslatef( center.x, center.y, center.z)
glRotatef( phi, 0.0, 0.0, 1.0)
renderBag()
```

Schwingung bei horizontalem Schlag

```
# Normierte Koordinate
pos = handPos.x / width

# Geschwindigkeit
v = 0.0
if( lastPos - pos >= 0.05):
    v = pos / dt
lastPos = pos

# Winkelgeschwindigkeit
omega_neu = omega + v
omega += dt * (-alpha * math.sin(phi) - beta * omega_neu)

# Winkel
phi = phi + dt * omega_neu
phi = phi / math.pi * 180.

# Translation/ Rotation
glTranslatef( center.x, center.y, center.z)
glRotatef( phi, 0.0, 0.0, 1.0)
renderBag()
```

Schwingung bei horizontalem Schlag

```
# Normierte Koordinate
pos = handPos.x / width

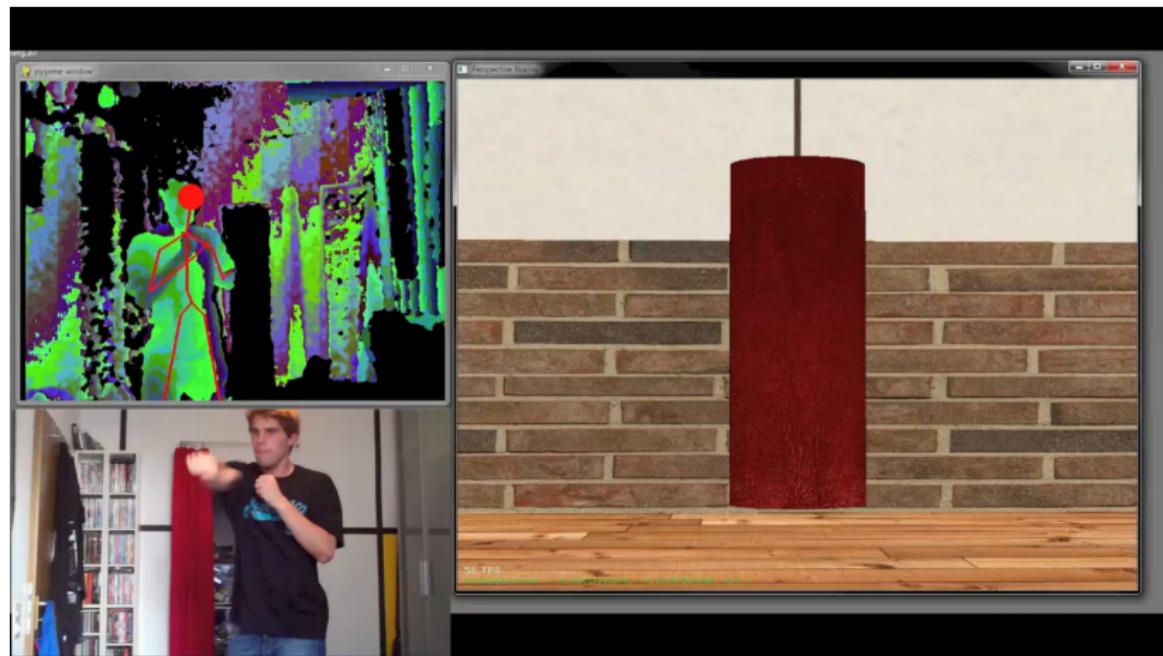
# Geschwindigkeit
v = 0.0
if( lastPos - pos >= 0.05):
    v = pos / dt
lastPos = pos

# Winkelgeschwindigkeit
omega_neu = omega + v
omega += dt * (-alpha * math.sin(phi) - beta * omega_neu)

# Winkel
phi = phi + dt * omega_neu
phi = phi / math.pi * 180.

# Translation/ Rotation
glTranslatef( center.x, center.y, center.z)
glRotatef( phi, 0.0, 0.0, 1.0)
renderBag()
```

Videodemonstration



Probleme

- Großer Rauschanteil im Tiefenbild ⇒ "Pseudoschläge"
- Genauigkeit der Tiefenabschätzung sehr gering
- Rauschen in Skelettdaten, unabhängig von Tiefeninformationen
- Zusammenarbeit zwischen pyGame (pyKinect) und OpenGL problematisch
- Geschwindigkeiteinbußen dank Python

Programm funktioniert nicht zufriedenstellend

3D-Effekt funktioniert besser als erwartet

Probleme

- Großer Rauschanteil im Tiefenbild ⇒ "Pseudoschläge"
- Genauigkeit der Tiefenabschätzung sehr gering
- Rauschen in Skelettdaten, unabhängig von Tiefeninformationen
- Zusammenarbeit zwischen pyGame (pyKinect) und OpenGL problematisch
- Geschwindigkeiteinbußen dank Python

Programm funktioniert nicht zufriedenstellend

3D-Effekt funktioniert besser als erwartet

Probleme

- Großer Rauschanteil im Tiefenbild ⇒ "Pseudoschläge"
- Genauigkeit der Tiefenabschätzung sehr gering
- Rauschen in Skelettdaten, unabhängig von Tiefeninformationen
- Zusammenarbeit zwischen pyGame (pyKinect) und OpenGL problematisch
- Geschwindigkeiteinbußen dank Python

Programm funktioniert nicht zufriedenstellend

3D-Effekt funktioniert besser als erwartet

Probleme

- Großer Rauschanteil im Tiefenbild \Rightarrow "Pseudoschläge"
- Genauigkeit der Tiefenabschätzung sehr gering
- Rauschen in Skelettdaten, unabhängig von Tiefeninformationen
- Zusammenarbeit zwischen pyGame (pyKinect) und OpenGL problematisch
- Geschwindigkeiteinbußen dank Python

Programm funktioniert nicht zufriedenstellend

3D-Effekt funktioniert besser als erwartet

Probleme

- Großer Rauschanteil im Tiefenbild \Rightarrow "Pseudoschläge"
- Genauigkeit der Tiefenabschätzung sehr gering
- Rauschen in Skelettdaten, unabhängig von Tiefeninformationen
- Zusammenarbeit zwischen pyGame (pyKinect) und OpenGL problematisch
- Geschwindigkeiteinbußen dank Python

Programm funktioniert nicht zufriedenstellend

3D-Effekt funktioniert besser als erwartet

Probleme

- Großer Rauschanteil im Tiefenbild \Rightarrow "Pseudoschläge"
- Genauigkeit der Tiefenabschätzung sehr gering
- Rauschen in Skelettdaten, unabhängig von Tiefeninformationen
- Zusammenarbeit zwischen pyGame (pyKinect) und OpenGL problematisch
- Geschwindigkeiteinbußen dank Python

Programm funktioniert nicht zufriedenstellend

3D-Effekt funktioniert besser als erwartet

Probleme

- Großer Rauschanteil im Tiefenbild ⇒ "Pseudoschläge"
- Genauigkeit der Tiefenabschätzung sehr gering
- Rauschen in Skelettdaten, unabhängig von Tiefeninformationen
- Zusammenarbeit zwischen pyGame (pyKinect) und OpenGL problematisch
- Geschwindigkeiteinbußen dank Python

Programm funktioniert nicht zufriedenstellend

3D-Effekt funktioniert besser als erwartet

- C++ und Microsoft eigenes SDK
 - Nur eine Grafik-API
 - Rauschverminderung durch Mittelung

- C++ und Microsoft eigenes SDK
- Nur eine Grafik-API
- Rauschverminderung durch Mittelung

- C++ und Microsoft eigenes SDK
- Nur eine Grafik-API
- Rauschverminderung durch Mittelung

Vielen Dank