



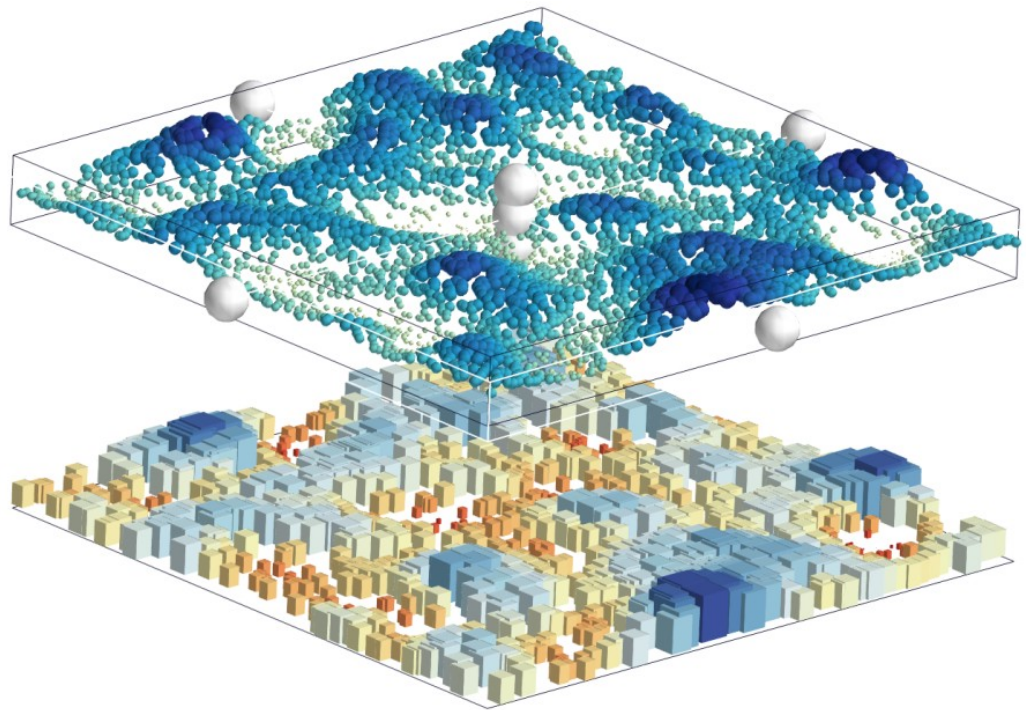
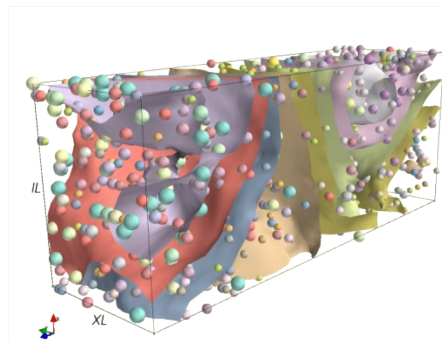
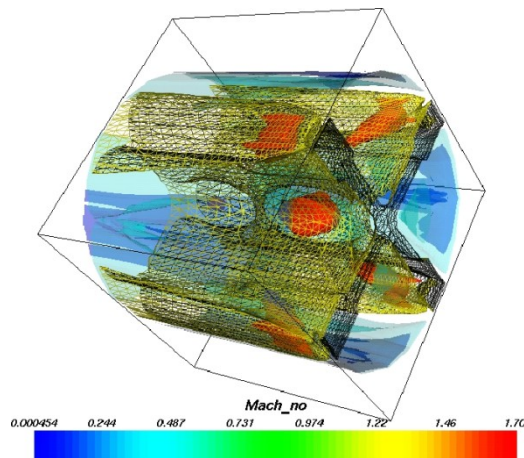
TECHNISCHE
UNIVERSITÄT
DRESDEN

Fakultät Maschinenwesen Institut für Verarbeitungsmaschinen und mobile Arbeitsmaschinen

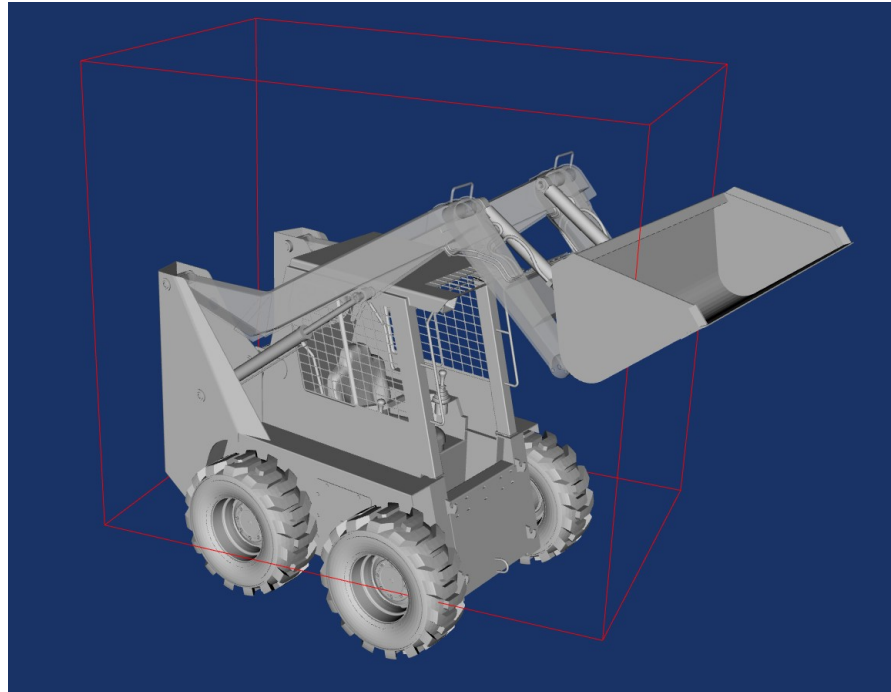
3D Visualisierung mit Python

13.05.2013 Sebastian Voigt

- 3D-Visualisierung von Messdaten, Simulationsergebnissen etc.
- Höhere Informationsdichte, anschaulichere Darstellungen



- 3D-Szenen mit Körpern, Beleuchtung, Animationen etc.
- Besseres Verständnis technischer Zusammenhänge
- Verwendung existierender Geometriemodelle



- PyOpenGL - <http://pyopengl.sourceforge.net/>
- Mayavi2 - <http://github.entthought.com/mayavi/mayavi/>
- **Vtk – The Visualisation Toolkit**- <http://www.vtk.org/>
- Vpython - <http://www.vpython.org/>
- Python-Ogre - <http://www.pythonogre.com/>
- Panda3D – <http://www.panda3d.org/>
- Wrapper für allerlei Scenegraphen, z.B. osgpython

Blender – teils in Python geschrieben, umfangreiche

Python Skripting API

Skript "interaktiv.py" öffnen

1. Geometrieprimitiv anlegen (Quader):

Quader

```
quaderSource = vtk.vtkCubeSource()  
quaderSource.SetXLength(0.3)  
quaderSource.SetYLength(0.1)  
quaderSource.SetZLength(0.1)
```

Mapper

```
quaderMapper = vtk.vtkPolyDataMapper()  
quaderMapper.SetInputConnection(quaderSource.GetOutputPort()  
)
```

Actor

```
quader = vtk.vtkLODActor()  
quader.SetMapper(quaderMapper)
```

2. Geometrieprimitiv anlegen (Kugel):

Kugel

```
kugelSource = vtk.vtkSphereSource()  
kugelSource.SetRadius(0.05)  
kugelSource.SetThetaResolution(10)    → Diskretisierung  
kugelSource.SetPhiResolution(10)
```

Mapper

```
kugelMapper = vtk.vtkPolyDataMapper()  
kugelMapper.SetInputConnection(kugelSource.GetOutputPort())
```

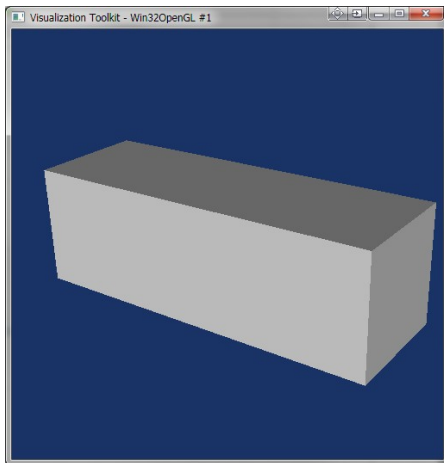
Actor

```
kugel = vtk.vtkLODActor()  
kugel.SetMapper(kugelMapper)
```

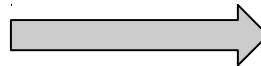
Actors in Renderer stecken:

```
ren.AddActor(quader)  
ren.AddActor(kugel)
```

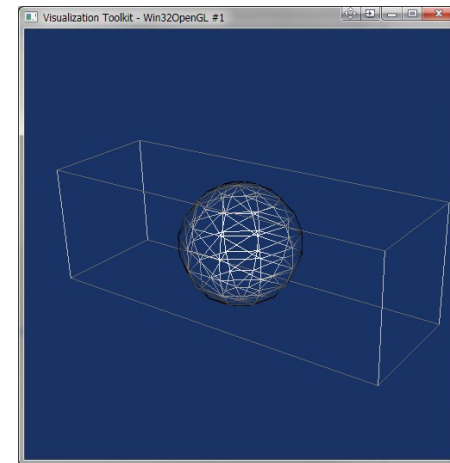
Ergebnis:



Taste <w>



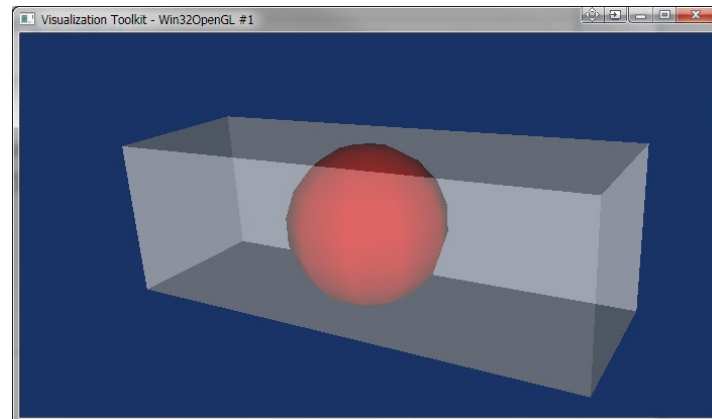
Taste <s>



Farbe und Transparenz anpassen:

```
kugel.GetProperty().SetColor(1,0,0)    # RGB Farbwerte 0..1  
quader.GetProperty().SetOpacity(0.5)
```

Ergebnis:



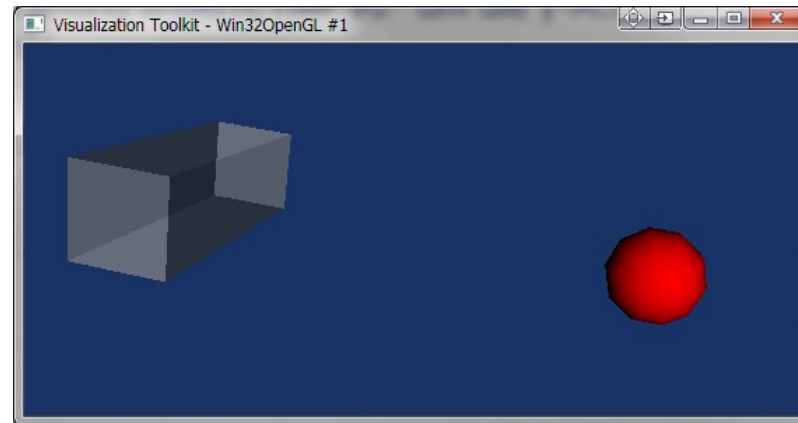
Position ändern (Verschiebung um x, y, z im globalen Koordinatensystem):

```
kugel.SetPosition(0.5,0,0)
```

Orientierung (Verdrehung ändern, hier 90° um die y-Achse im globalen Koordinatensystem):

```
quader.RotateWXYZ(90, 0,1,0)
```

Ergebnis:



Universelle Variante
via Poke-Matrix:

$$\text{poke} = \left[\begin{array}{ccc|c} \cdot & \cdot & \cdot & \cdot \\ \cdot & R & \cdot & p \\ \cdot & \cdot & \cdot & \cdot \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

R Konkret für unser Bsp.:

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

`R = eye(3)`

→ Einheitsmatrix (keine Rotation)

`p = array([1,0,0])`

→ Verschiebung um 1 in x-Richtung

`poke = vtk.vtkMatrix4x4()`

`for i in range(3):`

`for j in range(3):`

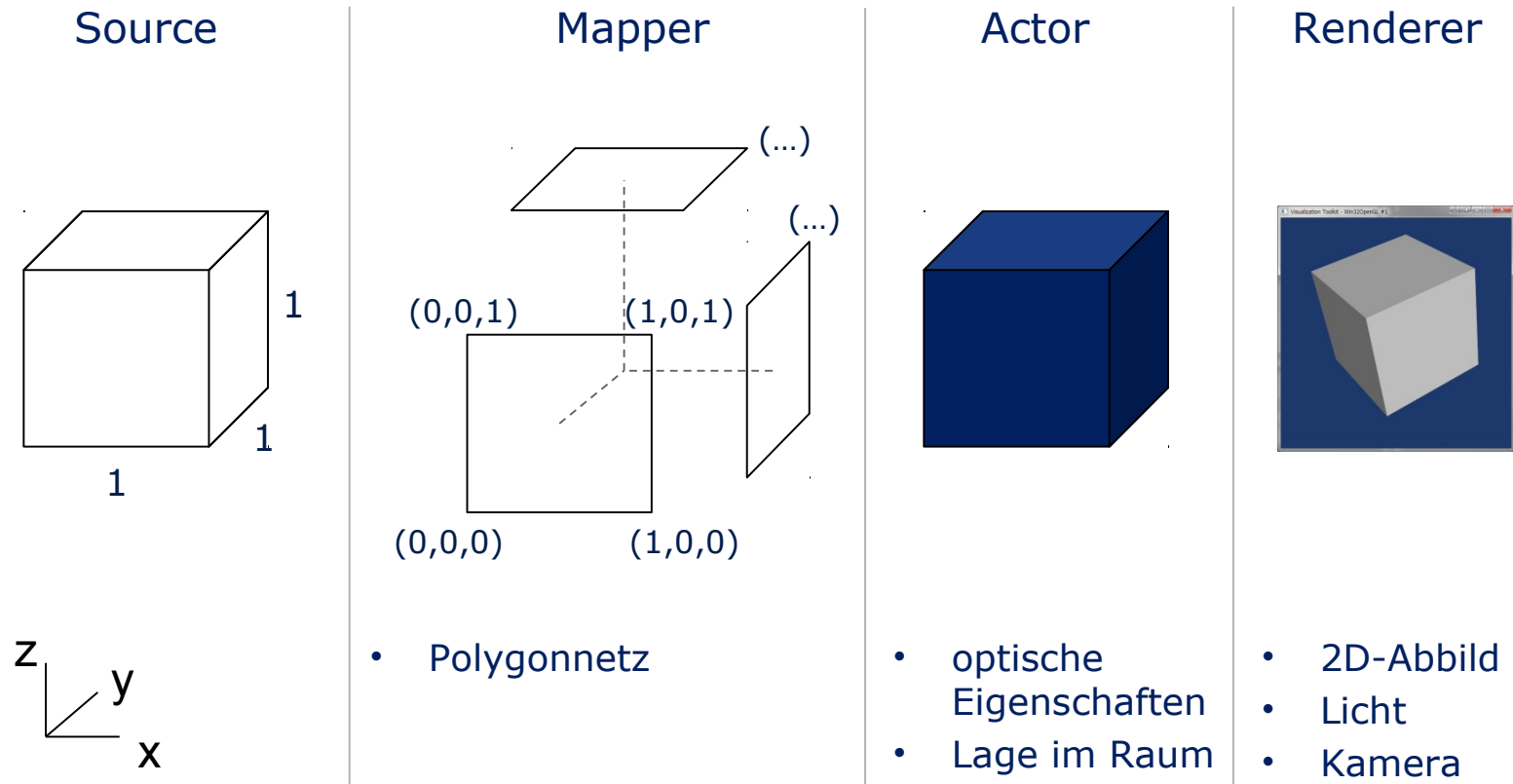
`poke.SetElement(i,j, R[i,j])`

`poke.SetElement(i,3, p[i])`

`actor.PokeMatrix(poke)`

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

Hintergrund zur Visualisierungspipeline:



Optische Eigenschaften werden über das Property-Objekt des actors eingestellt:

```
propObj = actor.GetProperty()
```

Transparenz: `propObj.SetOpacity(0.5)`

Farbe: `propObj.SetColor(0,0,1)` # RGB

Farbwerte 0..1

Reflexion: `propObj.SetAmbient(0.2)`

`propObj.SetDiffuse(0.8)`

`propObj.SetSpecular(0.5)`

`propObj.SetSpecularPower(0.5)`

`propObj.SetLighting(False)` # gut bei

vtkAxes

Kanten sichtbar: `propObj.SetEdgeVisibility(1)`

Größe: `propObj.SetScale(0.1)`

```
part = vtk.vtkCubeSource()  
part.SetXLength(10)  
part.SetYLength(1)  
part.SetZLength(1)
```

```
part = vtk.vtkSphereSource()  
part.SetRadius(5)  
part.SetThetaResolution(20)  
part.SetPhiResolution(20)
```

```
part = vtk.vtkLineSource()  
part.SetPoint1(0,0,0)  
part.SetPoint2(10,0,0)
```

```
part = vtk.vtkSTLReader()  
part.SetFileName('part.stl')
```

```
part = vtk.vtkTextSource()  
part.SetText('Hallo Welt')
```

```
part = vtk.vtkConeSource()  
part.SetHeight(10)  
part.SetRadius(5)  
part.SetResolution(20)  
part.SetAngle(30)  
part.Capping(2)
```

```
part = vtk.vtkCylinderSource()  
part.SetRadius(1)  
part.SetHeight(10)  
Part.SetResolution(20)
```

```
part = vtk.vtkAxes()  
part.SetScaleFactor(1)
```

Code in leeres Python Skript kopieren und ausführen (evtl. Code neu formatieren und Steuerzeichen entfernen)

1. Mayavi2 – interaktive Umgebung zur 3D-Visualisierung, einfaches Interface mit grundlegenden Plotbefehlen ähnlich zu Matlab und matplotlib

2. Vpython – einfache Umgebung zur Erstellung von 3D-Szenen und Simulationen

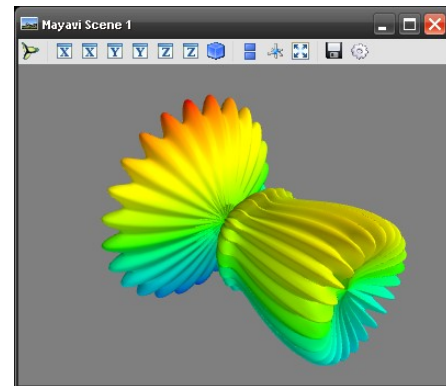
```
from numpy import pi, sin, cos, mgrid
from enthought.mayavi import mlab

# Daten erzeugen
dphi, dtheta = pi/250.0, pi/250.0
[phi,theta] = mgrid[0:pi+dphi*1.5:dphi,0:2*pi+dtheta*1.5:dtheta]

m0 = 4; m1 = 3; m2 = 2; m3 = 3; m4 = 6; m5 = 2; m6 = 6; m7 = 4;
r = sin(m0*phi)**m1+cos(m2*phi)**m3+sin(m4*theta)**m5+cos(m6*theta)**m7

x = r*sin(phi)*cos(theta)
y = r*cos(phi)
z = r*sin(phi)*sin(theta)

# anschauen
s = mlab.mesh(x, y, z)
mlab.show()
```



```
from visual import *

# Koerper anlegen
floor = box(pos=(0,0,0), length=4, height=0.5, width=4,
            color=color.blue)
ball = sphere(pos=(0,4,0), radius=1, color=color.red)

# Startwert fuer Geschwindigkeit und Schrittweite
ball.velocity = vector(0,-1,0)
dt = 0.01

# Simulation
while 1:
    rate(100)
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < ball.radius:
        ball.velocity.y = abs(ball.velocity.y)
    else:
        ball.velocity.y = ball.velocity.y - 9.81*dt
```

