TECHNISCHE UNIVERSITÄT DRESDEN

PYTHONKURS FÜR INGENIEUR:INNEN

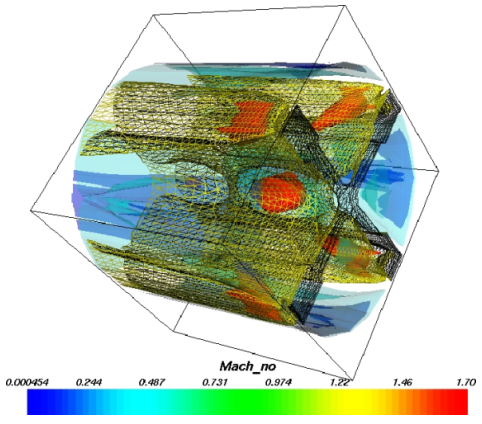
3D Visualisierung mit Python

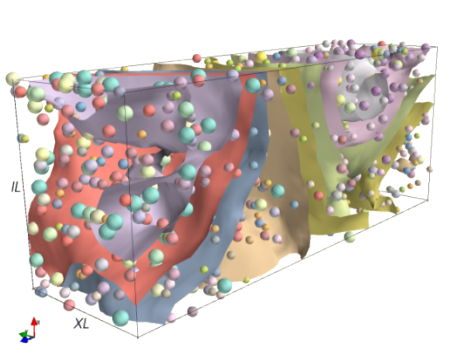
Sebastian Voigt, Carsten Knoll, TU Dresden

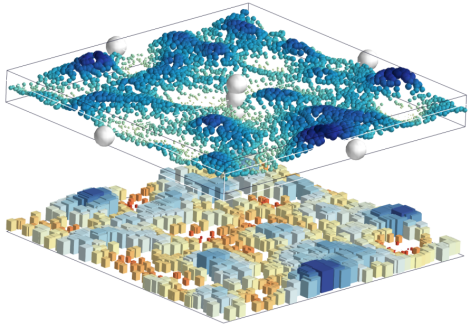
Dresden, 07.12.2020

Anwendungsgebiete I

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

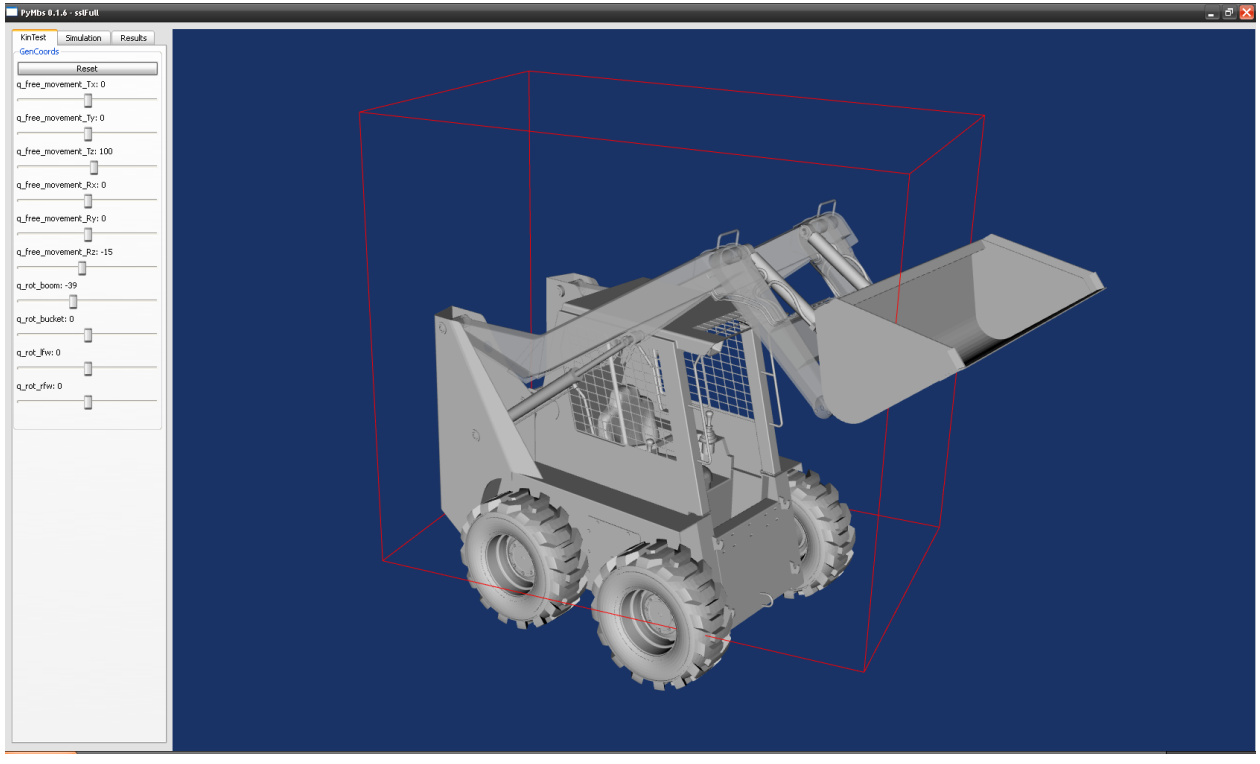






Anwendungsgebiete II

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



Python Pakete zur 3D-Visualisierung

- Vtk – The Visualisation Toolkit- `http://www.vtk.org/`
  - Wird im Kurs verwendet
  - Installation: `pip install vtk` (aus Anaconda-Prompt)

Was gibt es sonst noch?

- Mayavi2 - `http://github.enthought.com/mayavi/mayavi/`
- Vpython - `http://www.vpython.org/`

- PyOpenGL - `http://pyopengl.sourceforge.net/`
- Panda3D - `http://www.panda3d.org/`
- Blender - 3D-Modellierungssoftware mit guter Python-Integration `blender.org`
- ...

Einführung in `vtk`

- Geometrieprimitiv anlegen (Quader):
- Source → Mapper → Actor

Listing: vtk1.py:1-15

```
import vtk

# Quader anlegen (Source, Mapper, Actor)
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)
```

Einführung in `vtk`

- Geometrieprimitiv anlegen (Kugel):
- Source → Mapper → Actor

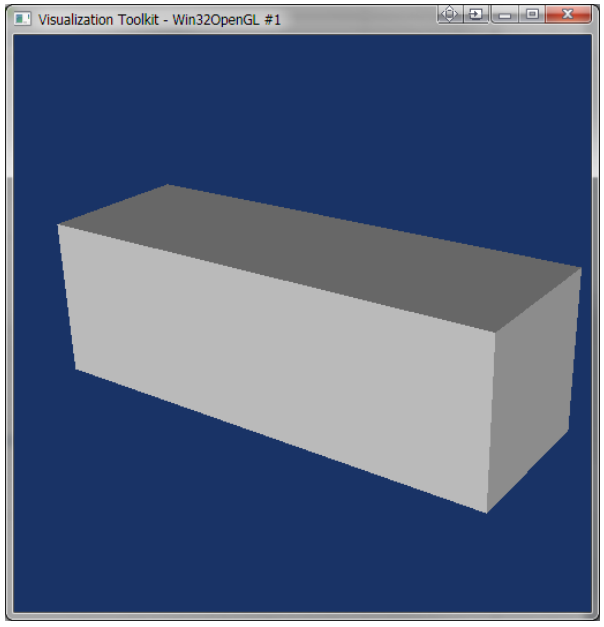
Listing: vtk1.py:17-29

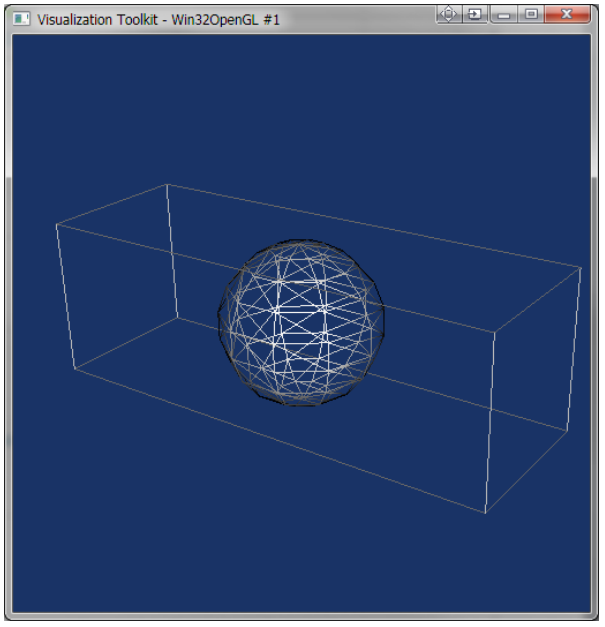
```
# Kugel anlegen (Source, Mapper, Actor)
kugelSource = vtk.vtkSphereSource()
kugelSource.SetRadius(0.08)
kugelSource.SetThetaResolution(10) # Diskretisierung
kugelSource.SetPhiResolution(10)

# Mapper
kugelMapper = vtk.vtkPolyDataMapper()
kugelMapper.SetInputConnection(kugelSource.GetOutputPort())

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

Einführung in vtk

Listing: vtk1.py:32-37

```
# Renderer
ren = vtk.vtkRenderer()

# Actors zum Renderer hinzufügen
ren.AddActor(quader)
ren.AddActor(kugel)
```





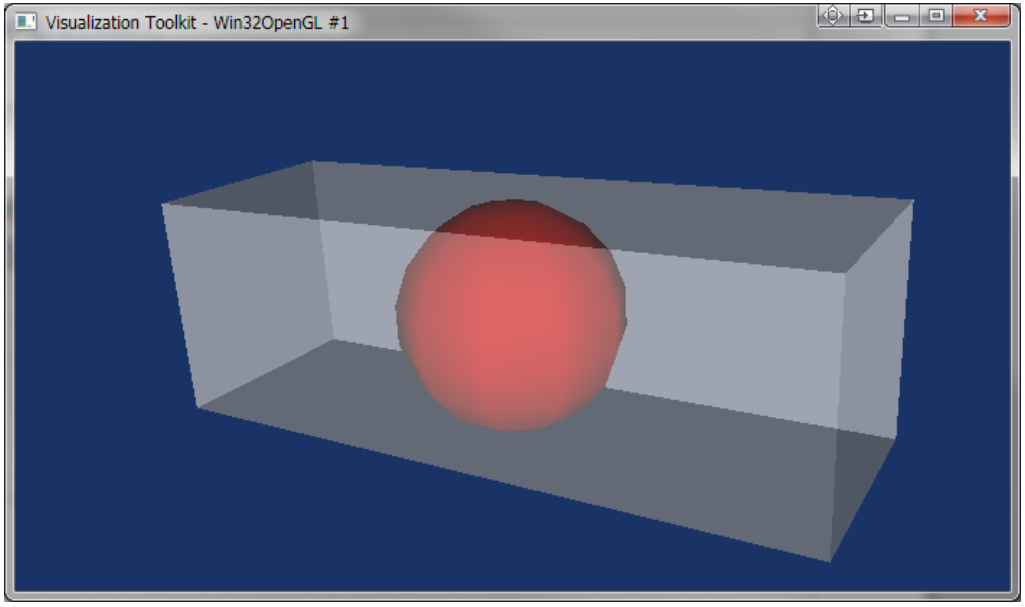
Umschalten der Ansichten mit W (wireframe) und S (solid).

Einführung in vtk

- Farbe und Transparenz anpassen:

```
import vtk.util.colors as colors
kugel.GetProperty().SetColor(colors.red)
quader.GetProperty().SetOpacity(0.5)
```

- Ergebnis:



vtk-Actors positionieren I

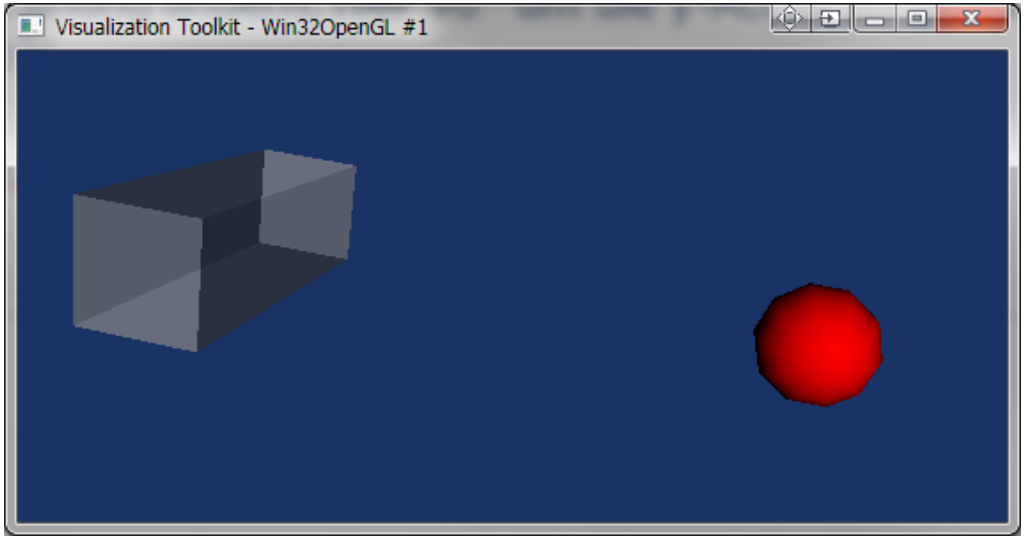
- Position ändern (Verschiebung um x, y, z im globalen Koordinatensystem):

```
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:



vtk-Actors positionieren II

- Universelle Variante via Poke-Matrix
- Aufbau: $P = \left(\begin{array}{ccc|c} \cdot & \cdot & \cdot & \vdots \\ \cdot & R & \cdot & r \\ \cdot & \cdot & \cdot & \vdots \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$, R Rotationsmatrix, r : Verschiebungsvektor

vtk-Actors positionieren II

- Universelle Variante via Poke-Matrix
- Aufbau: $P = \left(\begin{array}{ccc|c} \cdot & \cdot & \cdot & \vdots \\ \cdot & R & \cdot & r \\ \cdot & \cdot & \cdot & \vdots \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$, R Rotationsmatrix, r : Verschiebungsvektor

Bsp 1: Rotation um y-Achse:

$$R_{\mathrm{y}}(\varphi) = \begin{pmatrix} \cos\varphi & 0 & -\sin(\varphi) \\ 0 & 1 & 0 \\ \sin(\varphi) & 0 & \cos\varphi \end{pmatrix}$$

Bsp 2: Keine Rotation:

$$R = I_3$$

vtk-Actors positionieren II

- Universelle Variante via Poke-Matrix
- Aufbau: $P = \left(\begin{array}{ccc|c} \cdot & \cdot & \cdot & \vdots \\ \cdot & R & \cdot & r \\ \cdot & \cdot & \cdot & \vdots \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$, R Rotationsmatrix, r : Verschiebungsvektor

Listing: vtk2.py:58-65

```
P = np.eye(4)  # Einheitsmatrix (keine Rotation)
# Verschiebung: 1 in x-Richtung, 2 in y-Richtung usw
P[:-1, -1] = np.array([1, 2, 3])

# leere Matrix anlegen und Inhalt von T kopieren
poke_matrix = vtk.vtkMatrix4x4()
vtk.vtkMatrix4x4.DeepCopy(poke_matrix, P.flatten())
quader.PokeMatrix(poke_matrix)
```

Bsp 1: Rotation um y-Achse:

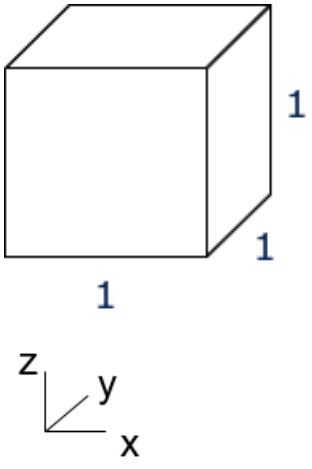
$$R_y(\varphi) = \begin{pmatrix} \cos\varphi & 0 & -\sin(\varphi) \\ 0 & 1 & 0 \\ \sin(\varphi) & 0 & \cos\varphi \end{pmatrix}$$

Bsp 2: Keine Rotation:

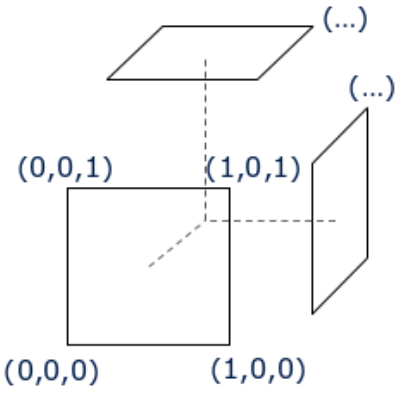
$$R = I_3$$

Hintergrund zur Visualisierungspipeline

Source

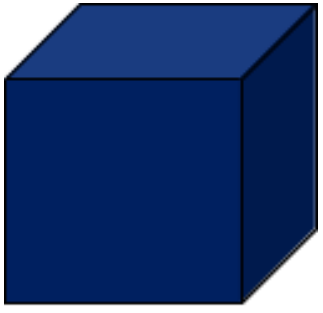


Mapper



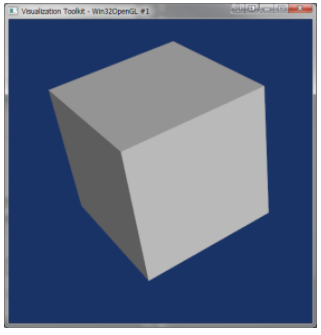
Polygonnetz

Actor



optische Eigenschaften, Lage im Raum

Renderer



2D-Abbild, Licht, Kamera

Eigenschaften des Actor-Objekts

- 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 oder vtk color`
- Reflexion:

```
propObj.SetAmbient(0.2)
propObj.SetDiffuse(0.8)
propObj.SetSpecular(0.5)
propObj.SetSpecularPower(0.5)
```

- Kanten sichtbar: `propObj.SetEdgeVisibility(1)`
- Größe: `propObj.SetScale(0.1)`

Kompliziertere Geometrien

- vtk $\neq$ 3D-CAD-Programm

$\rightarrow$ Kompliziertere Geometrien aus Datei laden

- Verbreitetes Dateiformat: STL (ursprünglich Stereo-Lithographie, siehe Wikipedia)

Kompliziertere Geometrien

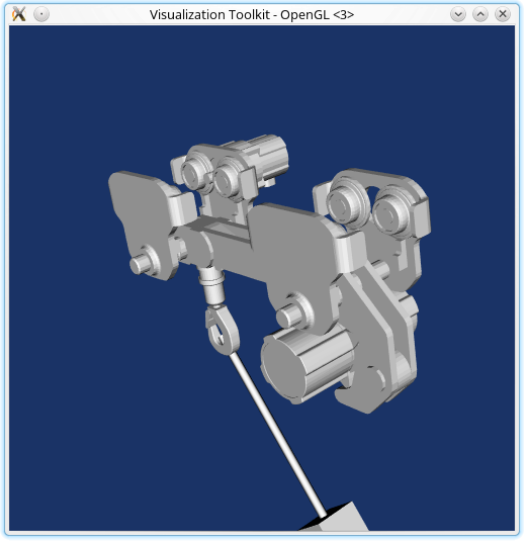
- vtk $\neq$ 3D-CAD-Programm

$\rightarrow$ Kompliziertere Geometrien aus Datei laden

- Verbreitetes Dateiformat: STL (ursprünglich Stereo-Lithographie, siehe Wikipedia)

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

Ergebnis:



Übersicht Geometrieprimitive

Kugel, Quader, Kegel, Zylinder, ...

```
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)
```