

Carsten Knoll

Chair of Fundamentals of Electrical Engineering

# Python for Engineers

## Pythonkurs für Ingenieur:innen

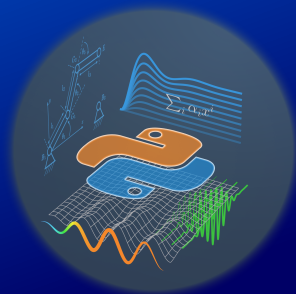
3D-Visualization (with vtk)

3D-Visualisierung (mit vtk)

Dresden (Online), 2024-12-20

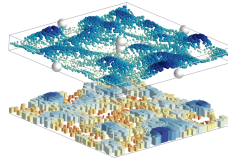
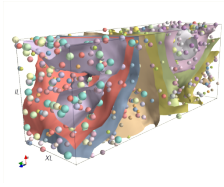
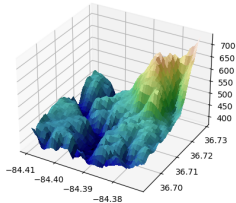
<https://tu-dresden.de/pythonkurs>

<https://python-fuer-ingenieure.de>



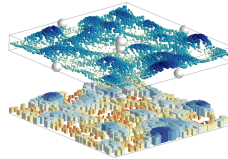
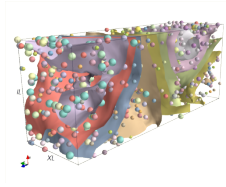
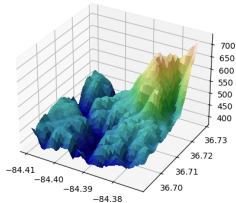
# Fields of Applications (1)

- 3D visualization of measurement data, simulation results, etc.  
→ higher information density, more illustrative representations



# Fields of Applications (1)

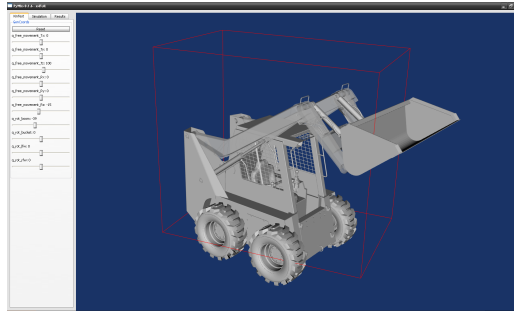
- 3D visualization of measurement data, simulation results, etc.  
→ higher information density, more illustrative representations



not covered here, but see e.g. [matplotlib 3d](#) and [Mayavi: 3D data visualization](#)

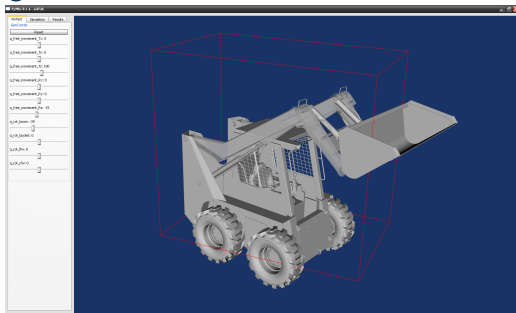
# Fields of Applications (2)

- 3D scenes with bodies, lighting, animations etc.
  - use of existing geometry models
- better understanding of technical relations



# Fields of Applications (2)

- 3D scenes with bodies, lighting, animations etc.
- use of existing geometry models
- better understanding of technical relations



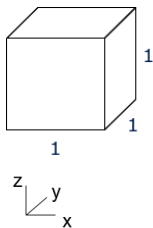
- we use: **vtk (visualisation toolkit)**, see <http://www.vtk.org/>
  - installation: `pip install vtk` (from anaconda prompt)
  - Python wrapper around C++ library

# vtk: Introduction by Example (1)

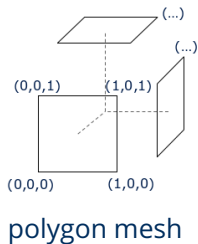
- create geometry primitive (here: cuboid):
- we need three objects (instances):
  - a source (geometric information)
  - a mapper (low level graphical representation; polygon mesh)
  - an actor (high level graphical representation; position, orientation, textures, ...)

# Background on the Visualization Pipeline

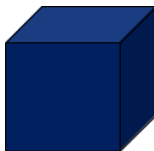
Source



Mapper

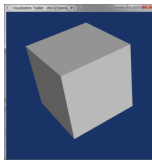


Actor



optical  
properties,  
placement  
in space  
(most  
important  
object)

Renderer



2D image,  
light,  
camera

# vtk: Introduction by Example (1)

- create geometry primitive (here: cuboid):
- we need three objects (instances):
  - a source (geometric information)
  - a mapper (low level graphical representation; polygon mesh)
  - an actor (high level graphical representation; position, orientation, textures, ...)

listing: example-code/vtk1.py

```
1 import vtk
2
3 # create cuboid source
4 my_cuboidSource = vtk.vtkCubeSource()
5 my_cuboidSource.SetXLength(0.3)
6 my_cuboidSource.SetYLength(0.1)
7 my_cuboidSource.SetZLength(0.1)
8
9 # create and connect cuboid mapper
10 my_cuboidMapper = vtk.vtkPolyDataMapper()
11 my_cuboidMapper.SetInputConnection(my_cuboidSource.GetOutputPort())
12
13 # create and connect cuboid actor
14 my_cuboid = vtk.vtkLODActor()
15 my_cuboid.SetMapper(my_cuboidMapper)
```



# vtk: Introduction by Example (2)

- create geometry primitive (here: cuboid):
- again: source, mapper, actor

listing: example-code/vtk1.py

```
17 # same (source, mapper, actor) for sphere
18 my_sphereSource = vtk.vtkSphereSource()
19 my_sphereSource.SetRadius(0.04)
20 my_sphereSource.SetThetaResolution(10) # discretization
21 my_sphereSource.SetPhiResolution(10)
22
23 # mapper
24 my_sphereMapper = vtk.vtkPolyDataMapper()
25 my_sphereMapper.SetInputConnection(my_sphereSource.GetOutputPort())
26
27 # actor
28 my_sphere = vtk.vtkLODActor()
29 my_sphere.SetMapper(my_sphereMapper)
```

# vtk: Introduction by Example (3)

displaying requires more objects:

- renderer (camera perspective, which object is behind which other, ...)
- render window (the 2D window to draw into)
- render window interactor (handle camera rotation, zoom (mouse and keyboard events))

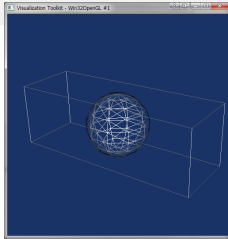
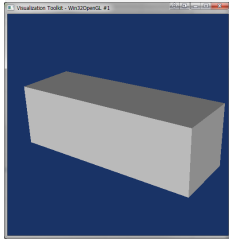
listing: example-code/vtk1.py

```
32 # create renderer
33 ren = vtk.vtkRenderer()
34
35 # add actors to renderer
36 ren.AddActor(my_cuboid)
37 ren.AddActor(my_sphere)
38
39 # create RenderWindow and RenderWindowInteractor
40 renWin = vtk.vtkRenderWindow()
41 renWin.AddRenderer(ren)
42 iren = vtk.vtkRenderWindowInteractor()
43 iren.SetRenderWindow(renWin)
```

# vtk: Introduction by Example (4)

listing: example-code/vtk1.py

```
45 # background color and window size
46 ren.SetBackground(0.1, 0.2, 0.4)
47 renWin.SetSize(800, 800)
48
49 # select a more convenient interactor style and initialize
50 iren.SetInteractorStyle(vtk.vtkInteractorStyleTrackballCamera())
51 iren.Initialize()
52
53 import vtk.util.colors as colors
54 my_sphere.GetProperty().SetColor(colors.red)
```



→ run `example-code/vtk1`

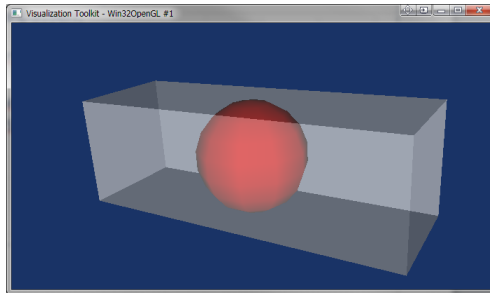
use mouse to zoom, rotate; switch views with `w` (wireframe) and `s` (solid).

# Introduction by Example (5)

- customize color and transparency (of actor objects):

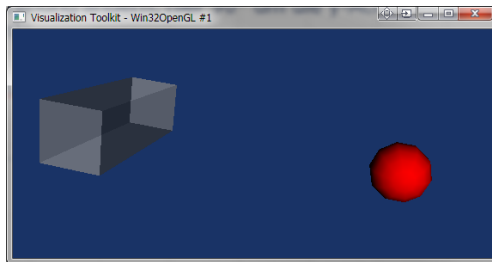
```
import vtk.util.colors as colors
my_sphere.GetProperty().SetColor(colors.red)
my_cuboid.GetProperty().SetOpacity(0.5)
```

- result:



# Positioning of vtk Actors (1)

- change position (shift by  $x, y, z$  in the global coordinate system):  
`my_sphere.SetPosition(0.5, 0, 0)`
- change orientation (rotate by  $90^\circ$  about the  $y$  axis in the global coordinate system):  
`my_cuboid.RotateWXYZ(90, 0, 1, 0)`
- result:



## Positioning of vtk Actors (2)

- universal transformations via poke matrix

- structure:  $P = \left( \begin{array}{ccc|c} \cdot & \cdot & \cdot & \vdots \\ \cdot & R & \cdot & r \\ \cdot & \cdot & \cdot & \vdots \\ \hline 0 & 0 & 0 & 1 \end{array} \right),$   $R : 3D \text{ rotation matrix},$   
 $r : 3D \text{ displacement vector}$

## Positioning of vtk Actors (2)

- universal transformations via pose matrix

- structure:  $P = \left( \begin{array}{ccc|c} \cdot & \cdot & \cdot & \vdots \\ \cdot & R & \cdot & r \\ \cdot & \cdot & \cdot & \vdots \\ \hline 0 & 0 & 0 & 1 \end{array} \right),$   $R$  : 3D rotation matrix,  
 $r$  : 3D displacement vector

example 1: rotation about  $y$  axis:

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

example 2: no rotation:

$$R = I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

## Positioning of vtk Actors (2)

- universal transformations via poke matrix

- structure:  $P = \left( \begin{array}{ccc|c} \cdot & \cdot & \cdot & \vdots \\ \cdot & R & \cdot & r \\ \cdot & \cdot & \cdot & \vdots \\ \hline 0 & 0 & 0 & 1 \end{array} \right),$   $R$  : 3D rotation matrix,  
 $r$  : 3D displacement vector

Listing: example-code/vtk2.py

```
62 P = np.eye(4) # 4x4 unit matrix as numpy array
63 P[:,-1] = np.array([1, 2, 2.8])
64
65 # vtk does not understand numpy datatypes directly
66 # create empty vtk matrix and copy values from P
67 poke_matrix = vtk.vtkMatrix4x4()
68 vtk.vtkMatrix4x4.DeepCopy(poke_matrix, P.flatten())
69 my_cuboid.PokeMatrix(poke_matrix)
```

example 1: rotation about  $y$  axis:

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

example 2: no rotation:

$$R = I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



# Properties of the Actor Object

- optical properties are set via the `Property` object of the actor:

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

- transparency: `propObj.SetOpacity(0.5)`

- color: `propObj.SetColor(0, 0, 1) # rgb`

- reflexion type:

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

- visibility of edges: `propObj.SetEdgeVisibility(1)`

- size: `propObj.SetScale(0.1)`

# More Complex Geometries

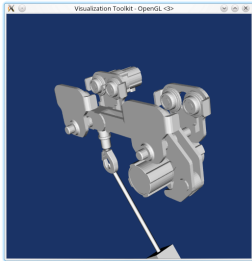
- $\text{vtk} \neq$  3D CAD software
- complex geometries should be loaded from files
- widespread file format: STL (originally for stereo lithography, see [Wikipedia](#))

# More Complex Geometries

- $\text{vtk} \neq$  3D CAD software
- complex geometries should be loaded from files
- widespread file format: STL (originally for stereo lithography, see [Wikipedia](#))

```
part = vtk.vtkSTLReader()  
part.SetFileName("exercise/data/trolley.stl")
```

result:



# Overview on Geometry Primitives

∃ source classes for: cuboid, sphere, line, cone, cylinder, ...

```
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.vtkCylinderSource()  
part.SetRadius(1)  
part.SetHeight(10)  
Part.SetResolution(20)  
  
part = vtk.vtkLineSource()  
part.SetPoint1(0,0,0)  
part.SetPoint2(10,0,0)
```

```
part = vtk.vtkConeSource()  
part.SetHeight(10)  
part.SetRadius(5)  
part.SetResolution(20)  
part.SetAngle(30)  
part.Capping(2)  
  
part = vtk.vtkAxes()  
part.SetScaleFactor(1)  
  
part = vtk.vtkSTLReader()  
part.SetFileName("part.stl")  
  
part = vtk.vtkTextSource()  
part.SetText("Hello World")
```

# Summary

- basics of 3D visualization with vtk
- visualization pipeline (source, mapper, actor, renderer, window, ...)
- actor objects are the most important
- placement via poke matrix
- geometry loading from stl files
- more information:
  - [vtk Python examples](#)
  - [vtk book](#)
  - [introduction to vtk and paraview](#) (lecture slides)