Python API: Customization and Visualization

1. TiGL Workshop, September 11 / 12, Cologne

Martin Siggel
German Aerospace Center

DLR

Outline

- How to modify TiGL-internal shapes
- Affine transformations (scaling, translations, rotations …)
- Boolean Operations
- Visualization with the Qt-based SimpleGui
- 3D rendering inside a Jupyter notebook
- Practical Session

Motivation

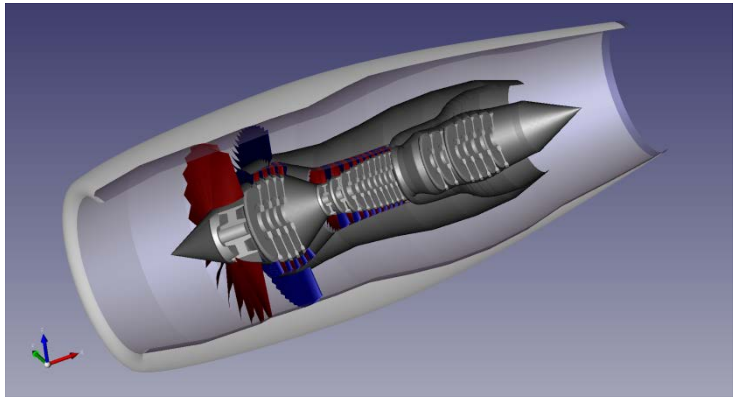
- Two use cases:

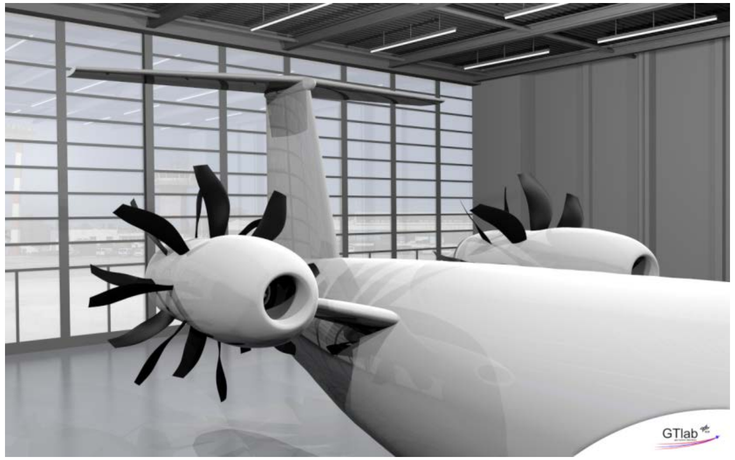
  1. Add new geometric components that are not included in TiGL

  2. Modify/improve existing components

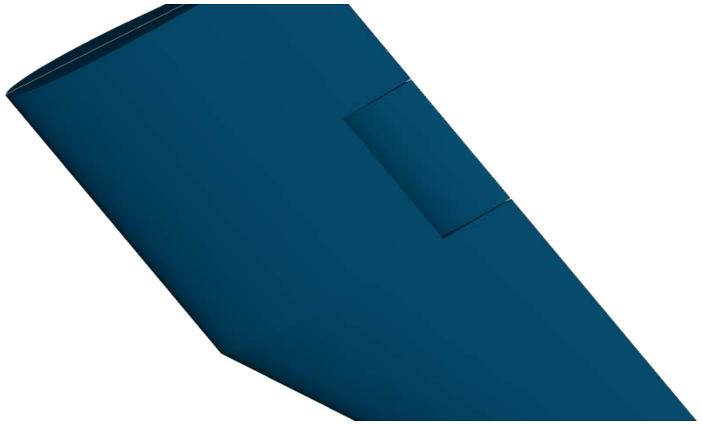
- First is straight forward: Just read out CPACS values and model your own geometry

- Second: How to modify the shapes? Can something happen?





Modification of TiGL shapes

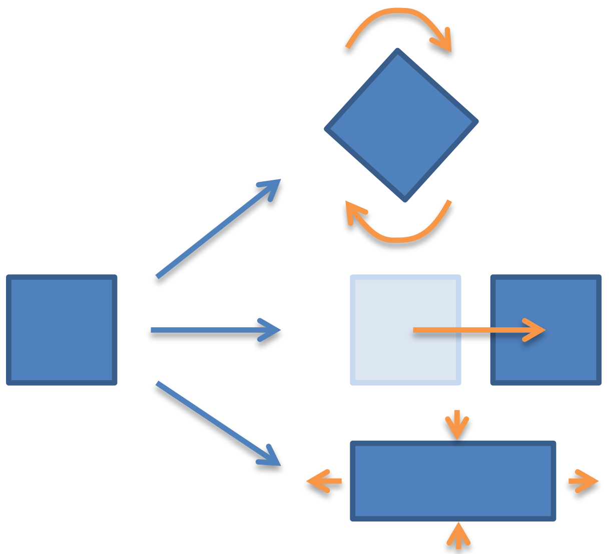


- Assume, you want to model wing flaps or wing caps
  - ➢ The wing shape has to be altered
- TiGL is not designed to change the internal shapes from outside
- TiGL is not designed to change the internal shapes from outside
- Still, this is possible!
- Each CNamedShape object has a `.Set(shape)` method:

```
# create new shape of modify the existing
new_shape = ...

# now set the changed loft to the wing
wing.get_loft().Set(new_shape)
```

Affine Transformations
How to move, resize, rotate shapes



- Shapes can be modified after creation
- Basic modification is affine transformation
- Use methods from class **tigl3.geometry.CTiglTransformation**. First build transformation matrix. Order matters!

Method	Description
`.add_translation(x, y, z)`	Move the shape
`.add_scaling(sx, sy, sz)`	Scale the shape along x, y, z axes
`.add_rotation_x(angle_degree)`	Rotate around the x axis
`.add_rotation_y(angle_degree)`	Rotate around the y axis
`.add_rotation_z(angle_degree)`	Rotate around the z axis
`.add_mirroring_at_xyplane()`	Mirror at the x-y plane
…	

- Transform the shape: `transformed_shape = trafo.transform(shape)`

Boolean Operations
The tigl3.boolean_ops module

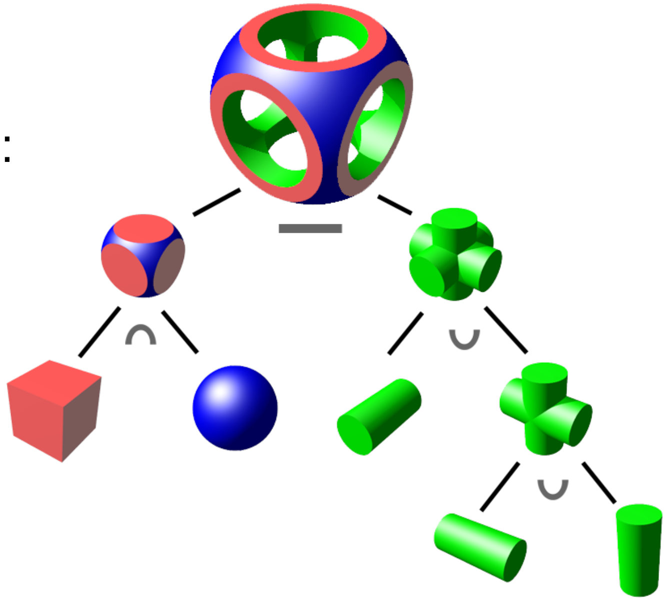
- Basic build blocks for constructive solid geometry

- Assume, we have two Shapes A and B. Typical Boolean Operations (BOPs) are:
    - Union: $A \cup B$
    - Difference: $A \setminus B$
    - Intersection: $A \cap B$

- Boolean Operations on B-Spline / NURBS are hard!
  Try to avoid them if possible.

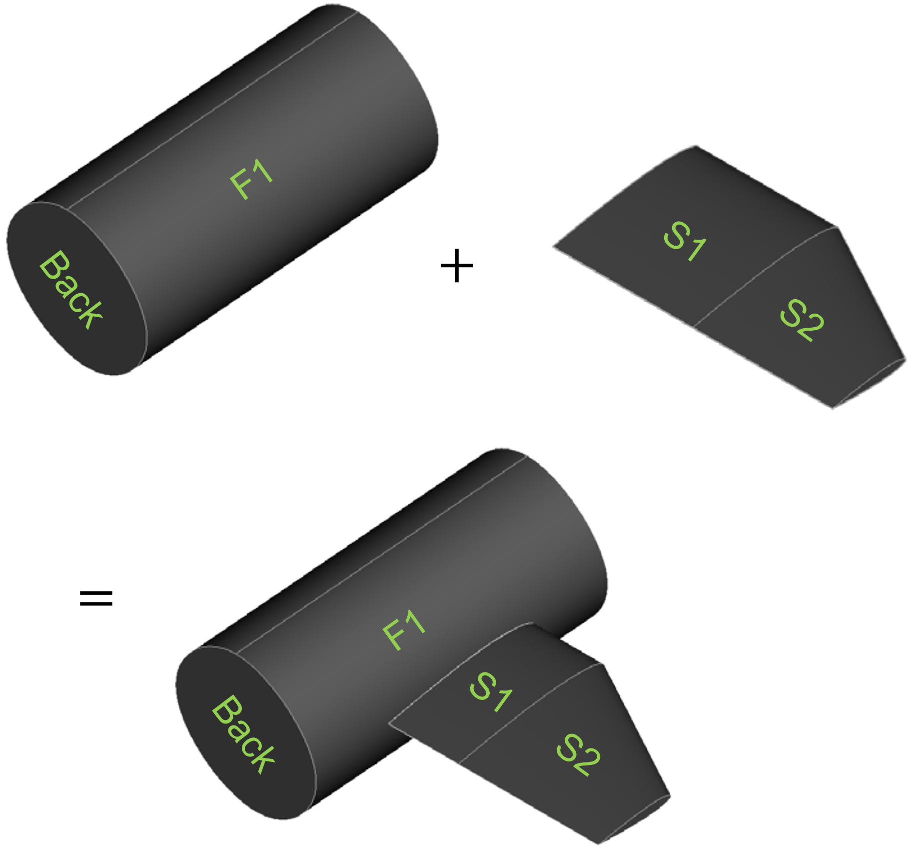
- OpenCASCADE offers BOPs, but:
    1. Unfortunately suffer from robustness issues
    2. Don‘t track shape modification (which face of a whole aircraft is from the wing?)

- TiGL BOPs wrap those from OpenCASCADE but add shape modification tracking!



Boolean Operations
The tigl3.boolean_ops module

- Faces are modified / trimmed by BOP
- Difficulty: Figure out, what face of the result is created from which input face
- TiGL BOPs do this for you!
    - Face names are assigned automatically by TiGL
    - TiGL keeps track of the CSG graph
- The following BOP classes from tigl3.boolean_ops can be used:
    - `CFuseShapes`: Boolean union of multiple shapes at once
    - `CMergeShapes`: Similar to CFuseShapes, but only for shapes that share adajcent faces
    - `CCutShape`: Boolean Difference
    - `CGroupShape`: No true BOP. Just a group of shapes.



Boolean Operations
Example

1. Lets cut away the internal part of the wing inside the fuselage:

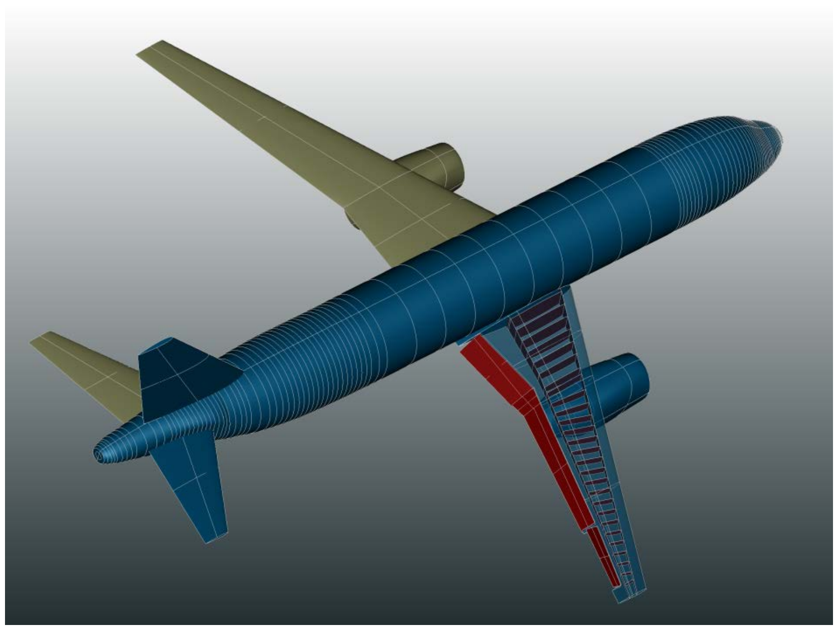
```
from tigl3.boolean_ops import CCutShape
cutted_wing = CCutShape(wing.get_loft(), fuselage.get_loft()).named_shape()
```

2. Now, lets fuse fuselage and both wings

```
fused_aircraft = CFuseShapes(fuselage.get_loft(),
                             [wing.get_loft(), wing.get_mirrored_loft()]).named_shape()
```

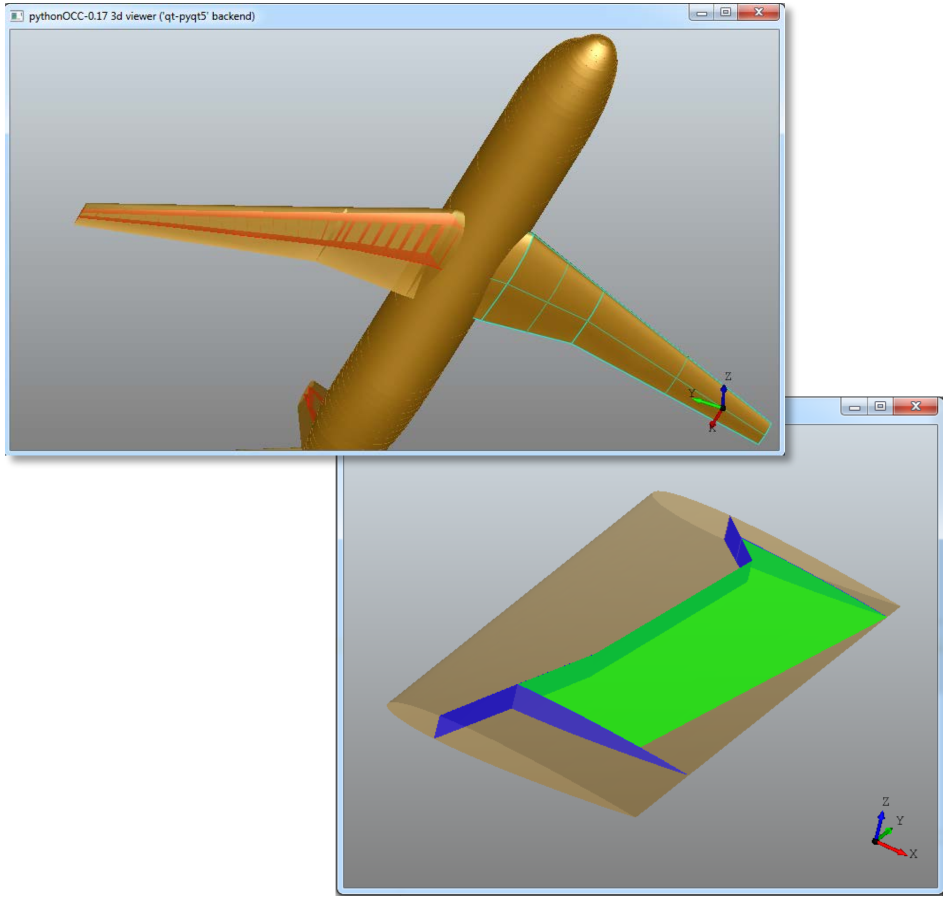
Visualization

- Nice images have often more impact than complicated algorithms!
- TiGL Viewer was initially developed only for debugging purposes!
- Visualization help debugging geometric algorithms or during modelling of complex shapes
- Good news:
    - PythonOCC comes with a 3D viewer, that can be also integrated into own user interfaces!
    - Also experimental renderer for Jupyter notebook



SimpleGui: A Qt-based 3D Viewer

- Draw OpenCASCADE shapes (TopoDS_Shape) with only a few lines of code
- Can be integrated in larger user interfaces
- Possible, to add callbacks to perform actions
- Features:
  - Selection of Colors
  - Transparency
  - Set material of shape
  - Draw Textures
  - Theoretically, also custom Shader code



SimpleGui: A Qt-based 3D Viewer

To open Viewer window and draw some shapes, we need 3 steps

1. Create the viewer and store it as viewer

```
from OCC.Display.SimpleGui import init_display
viewer, start_display, add_menu, add_function_to_menu = init_display()
```

2. Draw a shape. Notice, we must access the TopoDS_Shape from the CNamedShape!

```
viewer.DisplayShape(wing.get_loft().shape(), update=True)
```

   If update is True, the viewer will draw the shape immediately.

3. Start the event loop of the viewer to interact with the visualization:

```
start_display()
```

SimpleGui: A Qt-based 3D Viewer
More control

- The DisplayShape() method has several optional parameters to control transparency, color and texture:

```
DisplayShape(shapes, material=None,texture=None, color=None, transparency=None, update=False)
```

- Color can be
  - Either a string: e.g. „red“
  - A color value from the OCC.Quantity package: e.g. `OCC.Quantity.Quantity_NOC_GREEN`

- Material is of type **Graphic3d_NameOfMaterial** from OCC.Graphic3d:
  - `Graphic3d_NOM_CHROME, Graphic3d_NOM_ALUMINIUM, Graphic3d_NOM_METALIZED, Graphic3d_NOM_SHINY_PLASTIC, Graphic3d_NOM_STONE …`

- Texture: Why not try to figure it out?

SimpleGui: A Qt-based 3D Viewer
More control

- The viewer has many methods, which can be grouped as follows:
    - Mouse interaction
    - Selection of shapes
    - Modify eye + look-at position
    - Functions to add callbacks

- Find out, what the viewer can do by using the help

```
help(OCC.Display.OCCViewer)
```

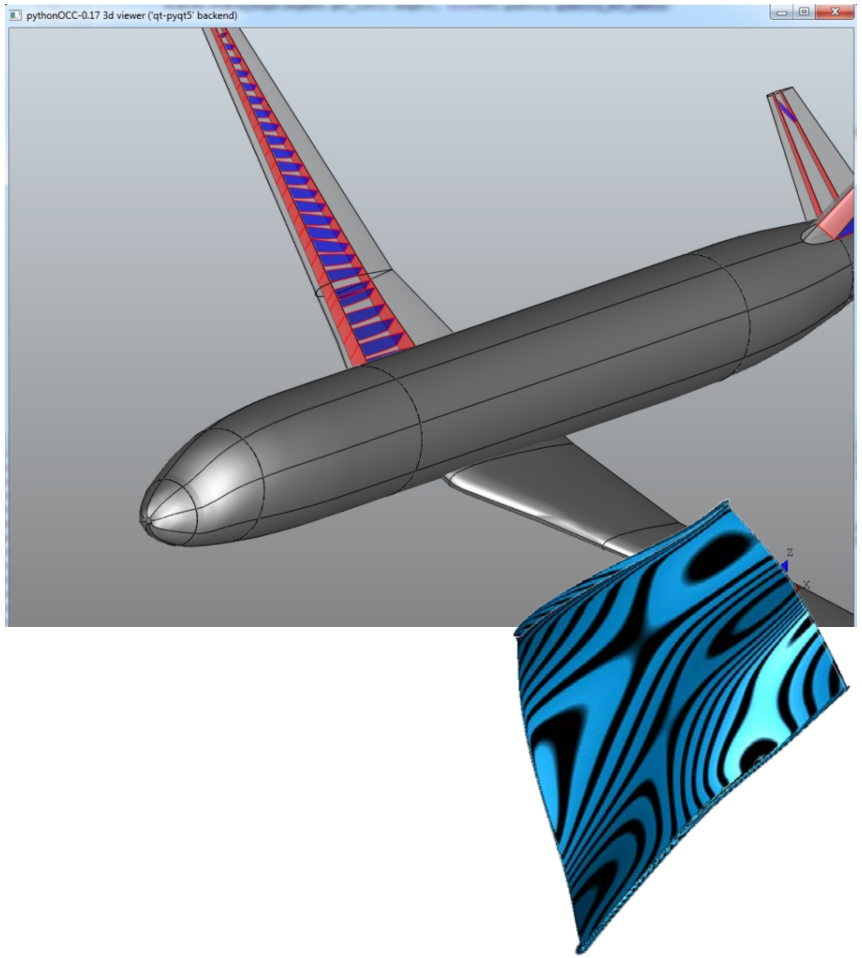
- Very useful command: Fit displayed objects to screen

```
viewer.FitAll()
```

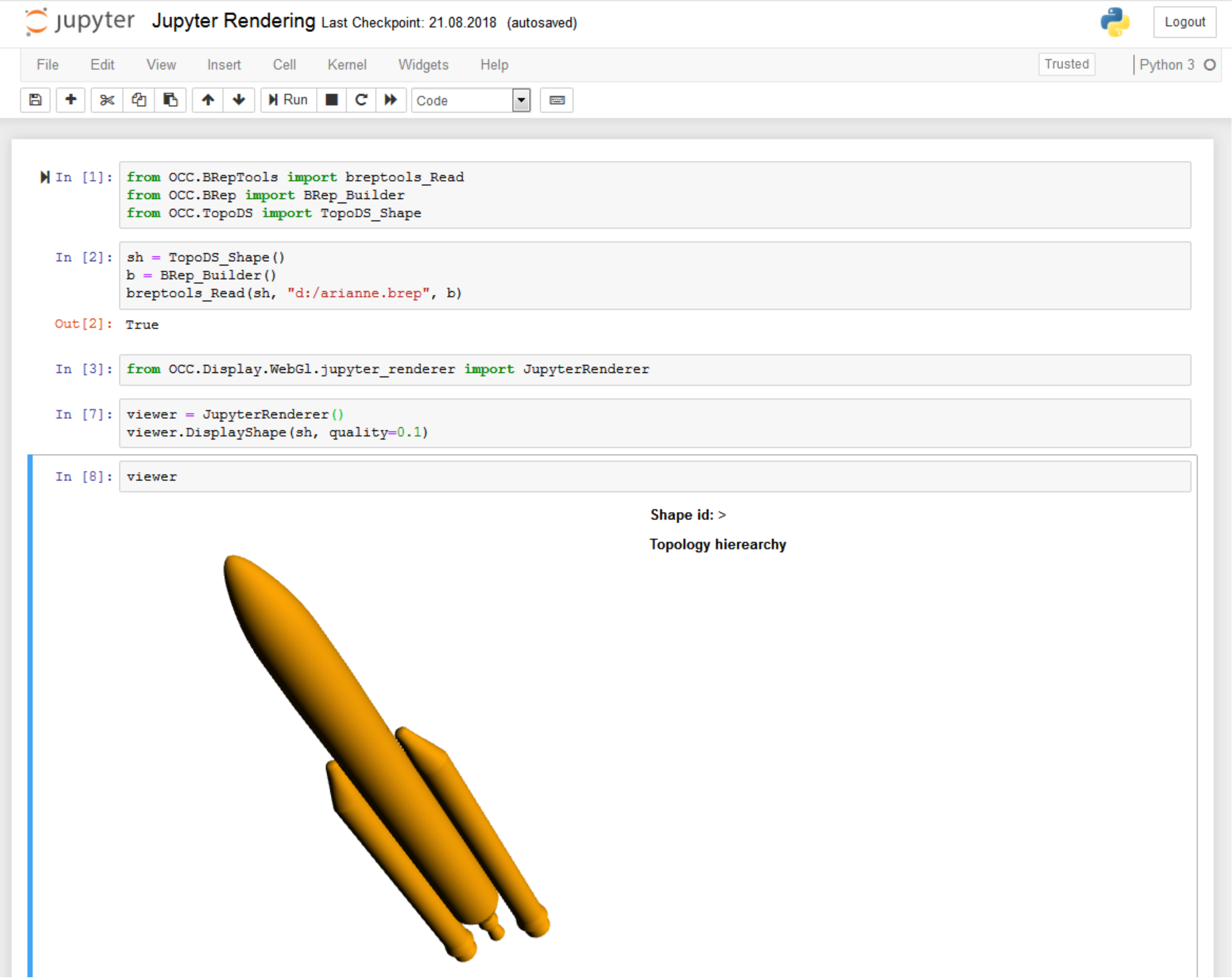
SimpleGui: A Qt-based 3D Viewer
Even more control

- Much more can be adjusted via **the Interactive Context** of the viewer.
- The context **manages** the **3D scene and all graphic attributes** (line colors, shading colors, custom shader code …)
- Access interactive context: `viewer.Context`
- Look into the OpenCASCADE documentation for much more control and customization:

  https://www.opencascade.com/doc/occt-7.3.0/overview/html/occt_user_guides__visualization.html



Visualization inside Jupyter Notebook



Visualization inside Jupyter Notebook

- Jupyter gives you a nice **interactive python shell** inside your browser

- Using **WebGL** and **Javascript**, it is possible to render 3D geometries on web pages

- The Jupyter renderer is
  - An **experimental** feature of pythonOCC
  - **Back-ported** into our conda packages from the latest source
  - Not as mature and has **less features** than the other viewer!

- Still, it is fun…

Visualization inside Jupyter Notebook
Howto

Again, we need 3 steps

1. Create the viewer

```
from OCC.Display.WebGl.jupyter_renderer import JupyterRenderer
viewer = JupyterRenderer()
```

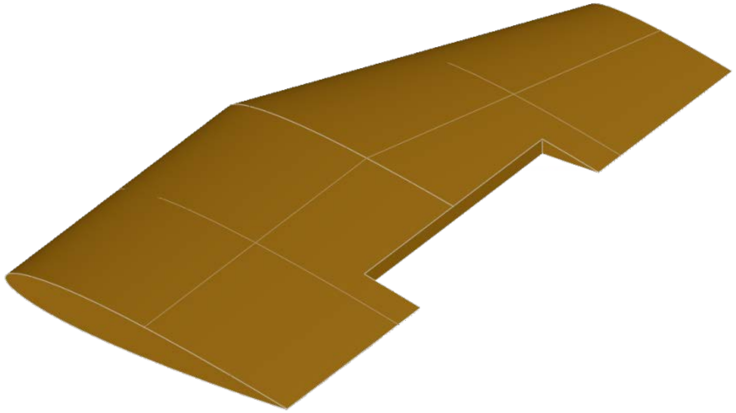
2. Add shapes to the viewer

```
viewer.DisplayShape(wing.get_loft().shape(), quality=0.1)
```

3. Render the viewer window

```
viewer
```

Practical Session: Customization Visualization



- Now it‘s your turn
- Goal: Model, export and visualize a wing flap cutout:
  You can do this, by subtracting a box from the wing.
- Possible Tasks:
  1. Open exercise 2 from course material inside a Jupyter notebook
  2. Access the first wing
  3. Create a box
  4. Move the box to the desired position
  5. Use TiGL’s Boolean operations to cut out the box
  6. Apply the result to the wing component
  7. Export the fused airplane to STEP format with: `tigl_handle.exportConfiguration`
  8. Visualize the result with the SimpleGui
  9. Try to change color etc
  10. Visualize the result directly in the Notebook

Any Questions?