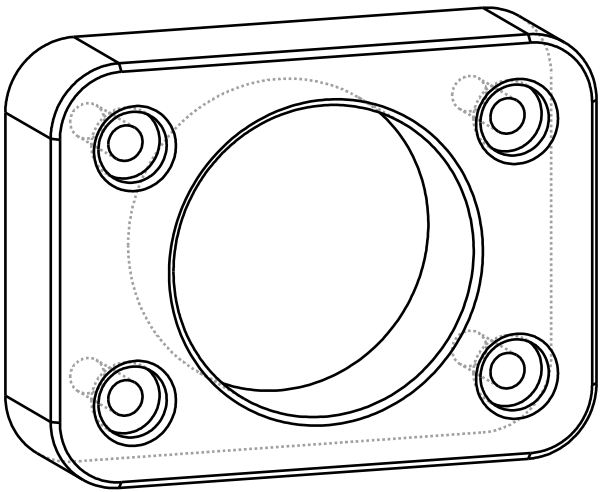
CadQuery Assembly System

Adam Urbańczyk

Introduction

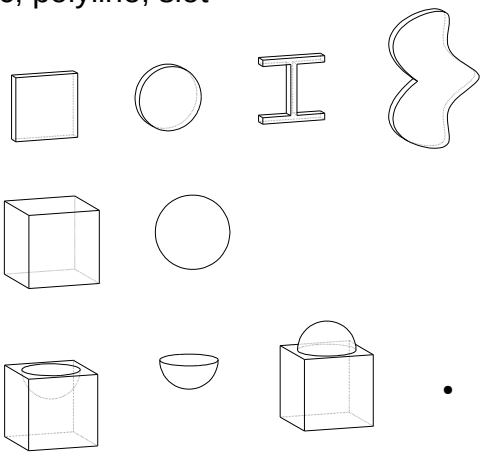
CadQuery is a Python module for building parametric 3D CAD models in boundary representation (B-rep)

```
import cadquery as cq

height = 40.0
width = 30.0
thickness = 10.0
radius = 11.0
padding = 12.0
rf = 5
cbore_r1,cbore_r2, cbore_d = 2.5, 5, 2
ch = .5

Result = (
    cq.Workplane()
    .rect(height, width).circle(radius).extrude(thickness)
    .faces(">Z").workplane()
    .rect(height - padding, width - padding, forConstruction=True)
    .vertices().cboreHole(cbore_r1,cbore_r2, cbore_d)
    .edges("|Z").fillet(rf)
    .faces('>Z').chamfer(ch)
)
```

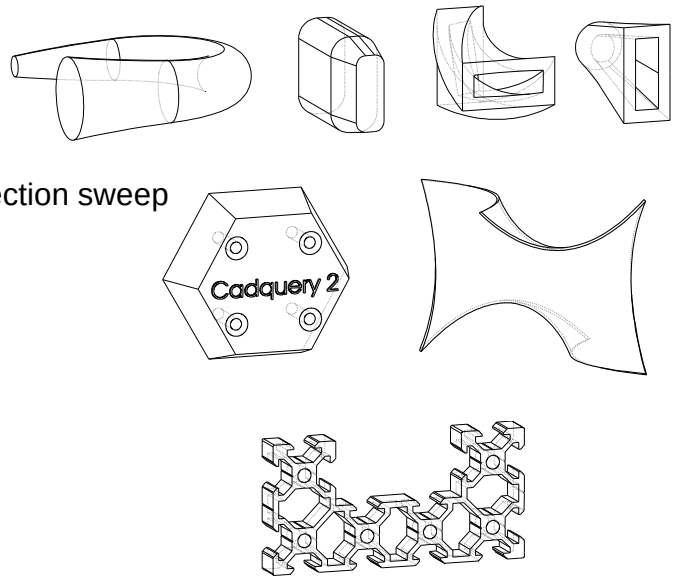


Capabilities

- 2D primitives
  - Rectangle, circle, ellipse, arc, polyline, slot
  - Spline
  - Parametric curves
  - Offset
- 3D primitives
  - Box, sphere
- CSG operations
  - Cut
  - Intersect
  - Union
- Selectors DSL
  - Choose vertices, edges, faces, solids
  - Combine selectors logically or chain them
  - Support tagging of elements



- 3D operations
  - Extrude (tapered, twisted)
  - Revolve
  - Loft
  - Shell
  - Fillet, chamfer
  - Sweep / multi-section sweep
  - 3D text
  - Fill
- Supported formats
  - STEP (R/W)
  - DXF (R/W)
  - STL (W)
  - AMF (W)
  - SVG (W)
  - VRML (W)



Whats new in CQ2.1

- Moved to OCCT 7.4 and custom Python bindings (OCP) using pybind11
- Most of the codebase has type annotations and is checked using MyPy
- DXF reading and writing support – including splines
- Tagging and additional selectors
- Assembly class
- Constraint solver for assemblies

CQ.Assembly

- Goals
  - Lightweigth
  - Encode color and location
  - Allow nesting
- Simple API
  - *add*
  - *constrain*
  - *solve*
  - *save*
- API is fluent, but does not keep history

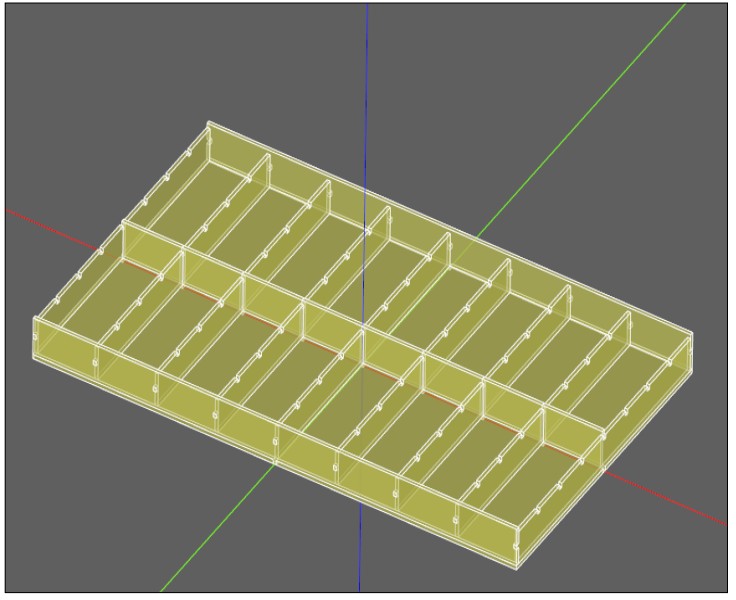
```
class Assembly(object):

    loc: Location
    name: str
    color: Optional[Color]
    metadata: Dict[str, Any]

    obj: Union[Shape, Workplane, None]

    parent: Optional[Assembly]
    children: List[Assembly]
```

Assembly example - manual

Manual placement of elements:

```
assy = (
    cq.Assembly(base,name='base',color=cq.Color(1,1,.4,0.5))
    .add(side_l,name='side_l',loc=Loc(Vec(-(W-d)/2,0,d)))
    .add(side_l,name='side_r',loc=Loc(Vec(+(W-d)/2,0,d)))
    .add(front_f,name='front_f',loc=Loc(Vec(0,-(H-d)/2,d)))
    .add(front_b,name='front_b',loc=Loc(Vec(0,(H-d)/2,d)))
    )
```

- Location of elements is relative to the parent location
- If not specified, color is inherited from the parent



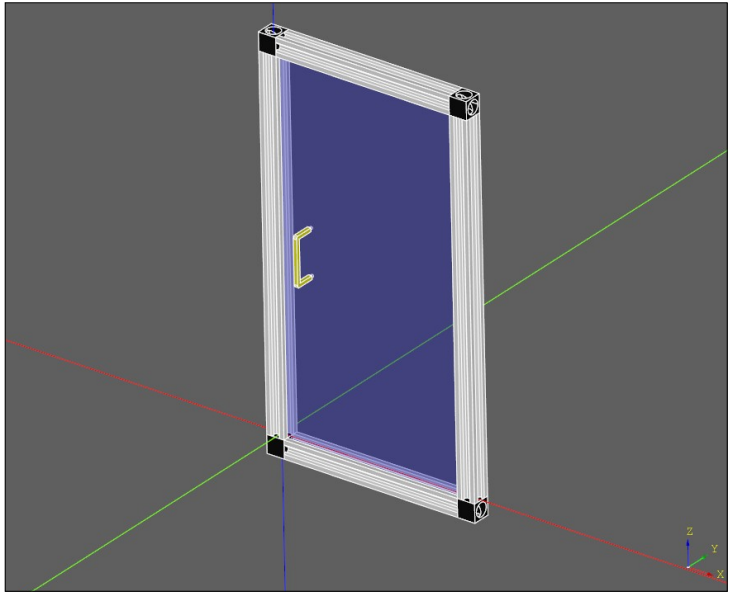
Full code here

Assembly example - constraints

Constraint based placement of elements:

```
(    door
    # left profile
    .constrain("left@faces@<Z", "con_bl?Z", "Plane")
    .constrain("left@faces@<X", "con_bl?X", "Axis")
    .constrain("left@faces@>Z", "con_tl?Z", "Plane")
    .constrain("left@faces@<X", "con_tl?X", "Axis")
)
```

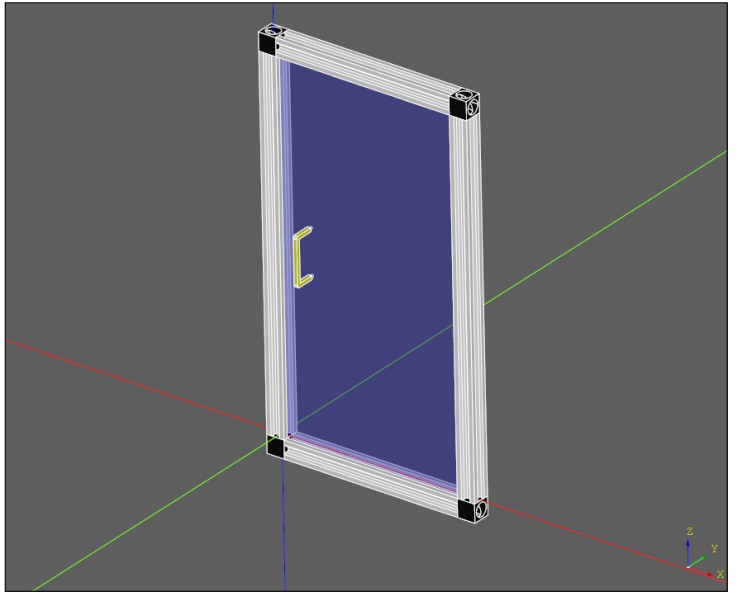
- Supported constraints
  - Axis
  - Point
  - Plane
- Entity selection
  - String selectors (including tags)
  - Explicit specification of a cq.Shape object



Full code here

Assembly example - constraints

```
(    door
    # left profile
    .constrain("left@faces@<Z", "con_bl?Z", "Plane")
    .constrain("left@faces@<X", "con_bl?X", "Axis")
    .constrain("left@faces@>Z", "con_tl?Z", "Plane")
    .constrain("left@faces@<X", "con_tl?X", "Axis")
    # top
    .constrain("top@faces@<Z", "con_tl?X", "Plane")
    .constrain("top@faces@<Y", "con_tl@faces@>Y", "Axis")
    # bottom
    .constrain("bottom@faces@<Y", "con_bl@faces@>Y", "Axis")
    .constrain("bottom@faces@>Z", "con_bl?X", "Plane")
    # right connectors
    .constrain("top@faces@>Z", "con_tr@faces@>X", "Plane")
    .constrain("bottom@faces@<Z", "con_br@faces@>X", "Plane")
    .constrain("left@faces@>Z", "con_tr?Z", "Axis")
    .constrain("left@faces@<Z", "con_br?Z", "Axis")
    # right profile
    .constrain("right@faces@>Z", "con_tr@faces@>Z", "Plane")
    .constrain("right@faces@<X", "left@faces@<X", "Axis")
    # panel
    .constrain("left@faces@>X[-4]", "panel@faces@<X", "Plane")
    .constrain("left@faces@>Z", "panel@faces@>Z", "Axis")
    # handle
    .constrain("panel?hole1", "handle?mate1", "Plane")
    .constrain("panel?hole2", "handle?mate2", "Point")
    .solve()
)
```



Full code here

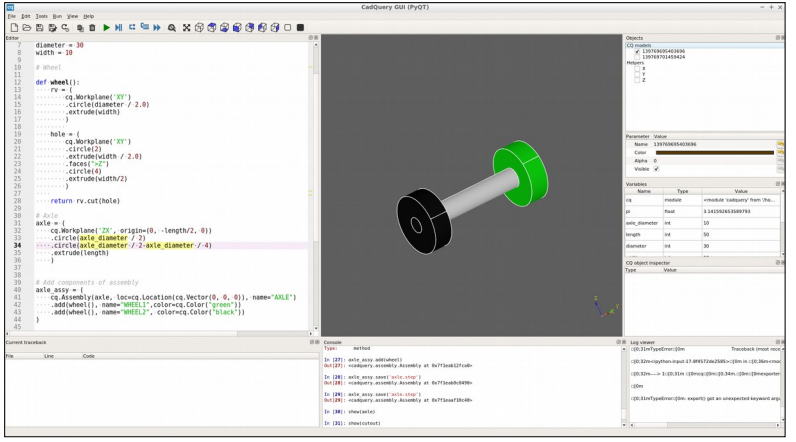
Exporting

Assemblies can be exported to STEP or XML

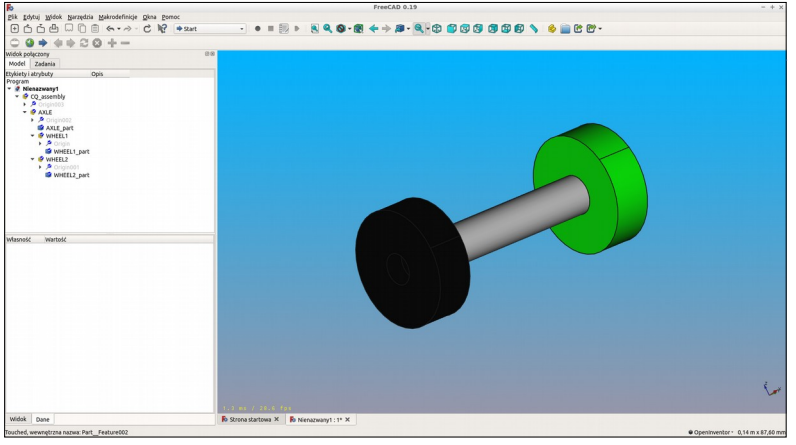
```
axle_assy.save('axle.step')
axle_assy.save('axle.xml')
```

- STEP is the main target
  - Exported STEP structure follows the assembly structure
  - Colors are saved
- „XML” is one of the internal OCCT formats
  - Meant for integrating with other tools using OCCT

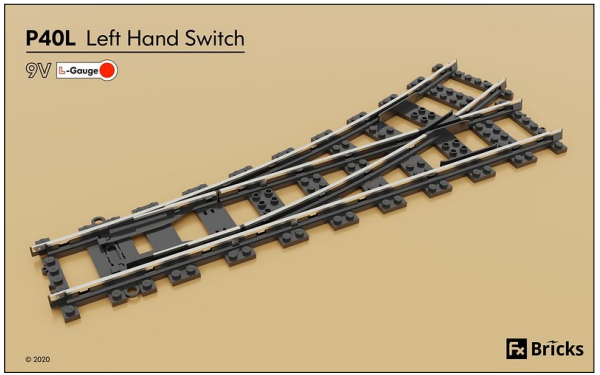
CQ-editor



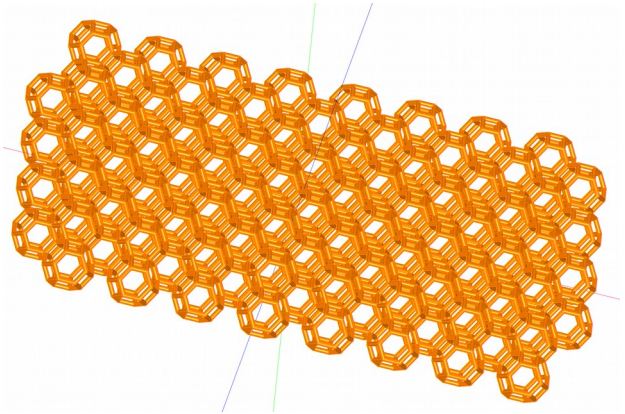
FreeCAD



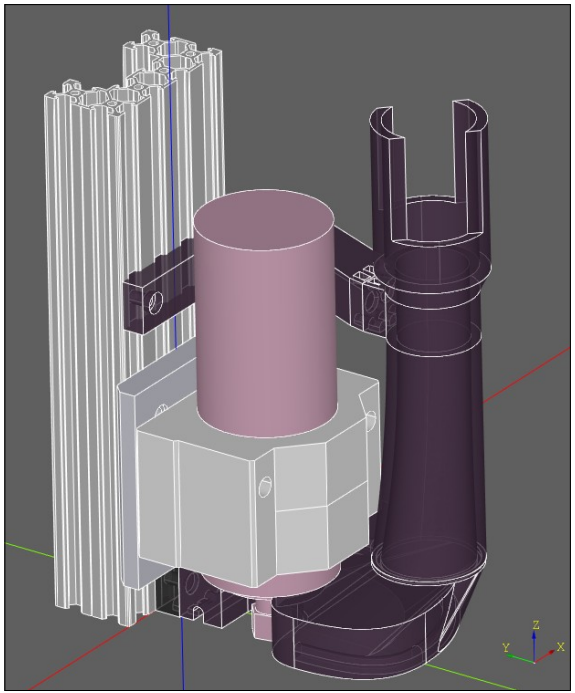
User showcase



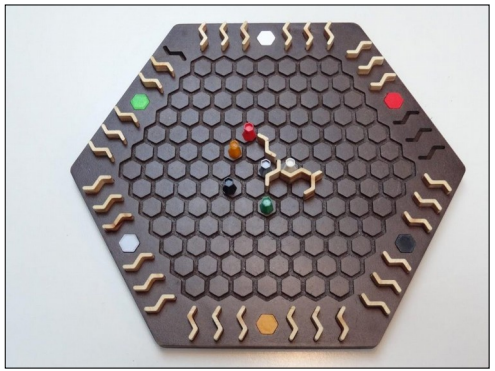
@michaelgale fx-bricks



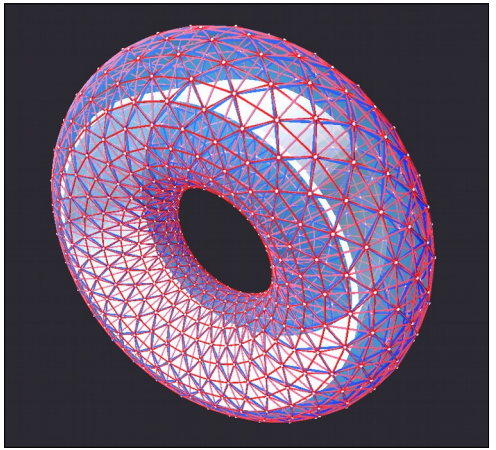
@bragostin CadQuery-Examples



@marcus7070 spindle-assy



Hexidor



@jmwright IslandOne

Future plans

- Move to OCCT 7.5
- Assembly improvements
  - More constraints
  - Speed
- Sketch class and sketch constraints
- glTF export
- VTK export

Standing on the shoulders of giants

CQ and CQ-editor wouldn’t be possible without the following open source projects

- Python
- OpenCascade
- FreeCAD
- PythonOCC
- PyParsing
- Conda
- PyOCCT
- Qt/PyQt
- Spyder
- PyQtGraph
- PyInstaller
- EzDXF
- Pybind11
- SciPy

```
import cadquery as cq

res = (
    cq.Workplane()
    .text('Questions?',10,2)
    .faces('>Z')
    .chamfer(.8,.2)
)
```