CNC Club
Vismach Tutorial

Acknowledgement:
Mogens Groth Nicolaisen
For Original uStepper 3D Robot Design

Capabilities of Vismach

- Visualise machine
- Finalise machine design
- Test machine kinematics
- Preview G-code instructions to avoid collisions
- Etc etc

Required Software

- Vismach is completely included in standard LinuxCNC installation

Using Vismach with LinuxCNC

- Small addition to .hal file
- Create .STL images of machine components (optional – but advised)
- Write script to define machine
- Add HAL animations to script
- Install/write the required kinematics software (if necessary - not included in this tutorial)

Additions to .hal file

loadusr -W ./CNCClubVismach

Name of Vismach script

Your Chosen Names

net j0 joint.0.pos-fb => CNCClubVismach.jointX

net j1 joint.1.pos-fb => CNCClubVismach.jointY

net j2 joint.2.pos-fb => CNCClubVismach.jointZ

LinuxCNC HAL pin Names (do not touch)

(found using Halscope)

LinuxCNC HAL signal Names (do not touch)

Matching code in Vismach Script

Name of Vismach script

```
halComponent = hal.component("CNCClubVismach")
halComponent.newpin("jointX", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.newpin("jointY", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.newpin("jointZ", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.ready()
```

Your Chosen Names

Any name – but “halComponent” or “halComp” or “halC” is recommended

“Minimum” Vismach Script

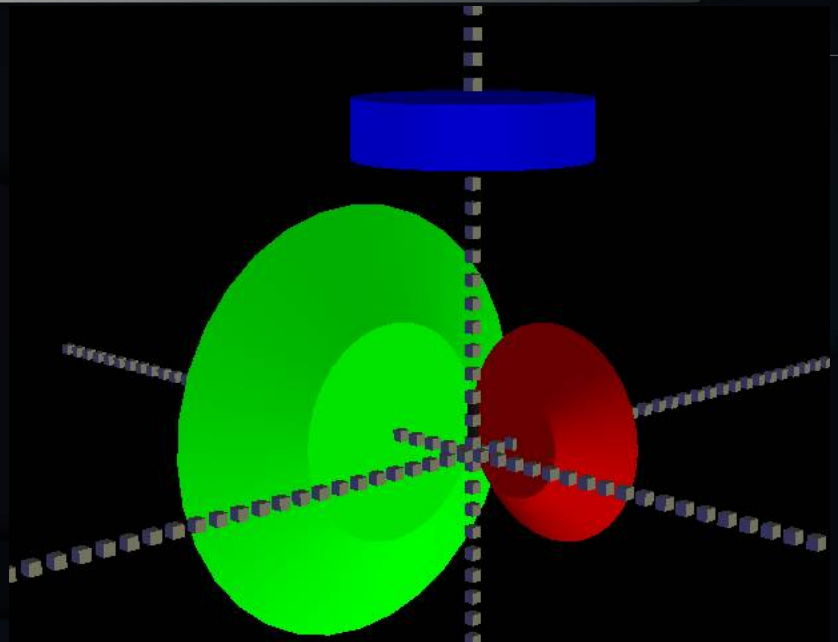
```
#!/usr/bin/python2
from vismach import *
import hal
import math
import sys

halComponent = hal.component("CNCClubVismach")
halComponent.newpin("jointX", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.newpin("jointY", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.newpin("jointZ", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.ready()
...
tooltip = Capture()
...
work = Capture()
...
main(model , tooltip , work , 1000)
```

Zoom Size

The Body of a Vismach Script

- Step 1: Make a model of your machine
- Step 2: Animate the model with the signals you get from the **HAL**

Making the model: Cylinders

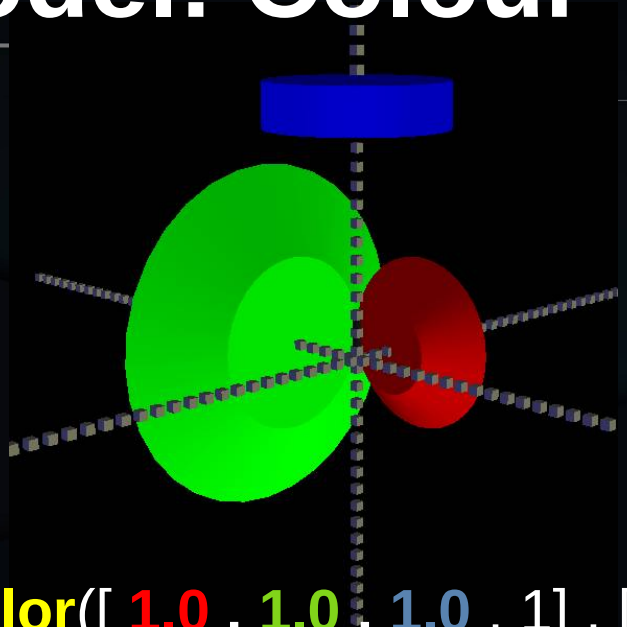


<<NameOfObject>> = **CylinderX**(<<xbeg>>,<<radiusbeg>>,<<xend>>,<<radiusend>>)

<<NameOfObject>> = **CylinderY**(<<ybeg>>,<<radiusbeg>>,<<yend>>,<<radiusend>>)

<<NameOfObject>> = **CylinderZ**(<<zbeg>>,<<radiusbeg>>,<<zend>>,<<radiusend>>)

Making the model: Colour



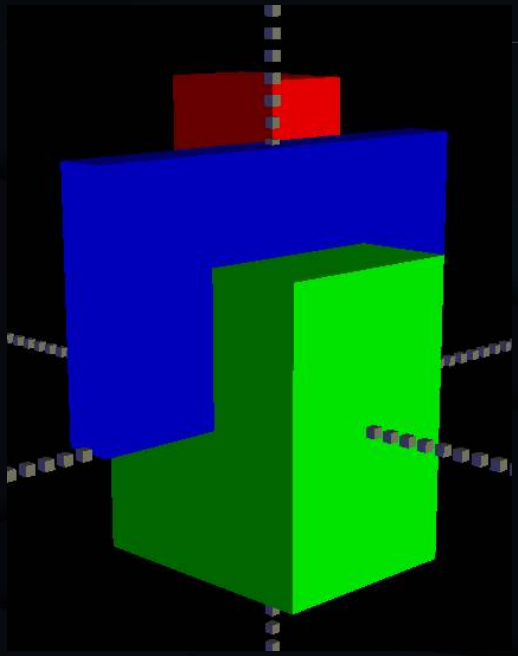
<<NameOfObject>> = **Color**([**1.0** , **1.0** , **1.0** , 1] , [<<NameOfObject>>])

% Red

% Green

% Blue

Making the model: Boxes

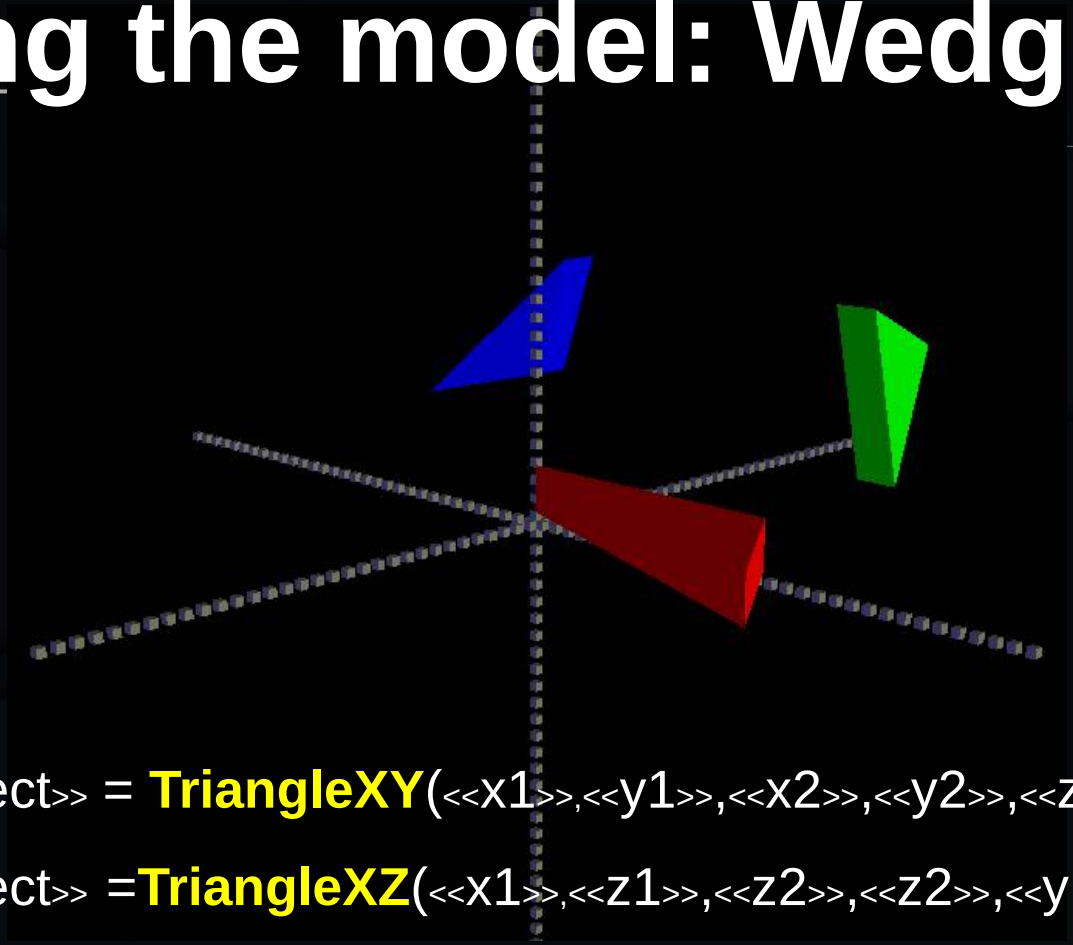


<<NameOfObject>> = **Box**(<<x1>>,<<y1>>,<<z1>>,<<x2>>,<<y2>>,<<z2>>)

<<NameOfObject>> = **BoxCentered**(<<xwidth>>,<<ywidth>>,<<zwidth>>)

<<NameOfObject>> = **BoxZ**(<<xwidth>>,<<ywidth>>,<<z>>)

Making the model: Wedges



<<NameOfObject>> = **TriangleXY**(<<x1>>,<<y1>>,<<x2>>,<<y2>>,<<z1>>,<<z2>>)

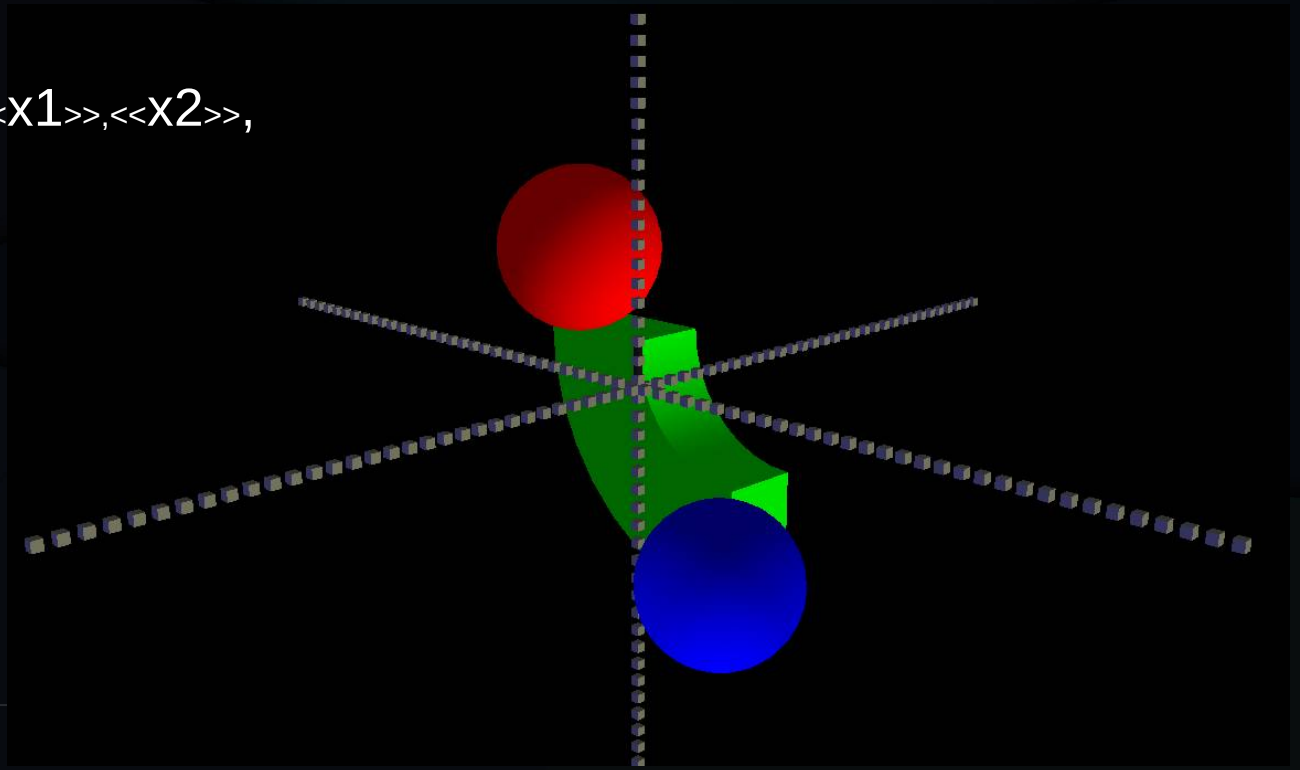
<<NameOfObject>> =**TriangleXZ**(<<x1>>,<<z1>>,<<z2>>,<<z2>>,<<y1>>,<<y2>>)

<<NameOfObject>> = **TriangleYZ**(<<y1>>,<<z1>>,<<y2>>,<<z2>>,<<x1>>,<<x2>>)

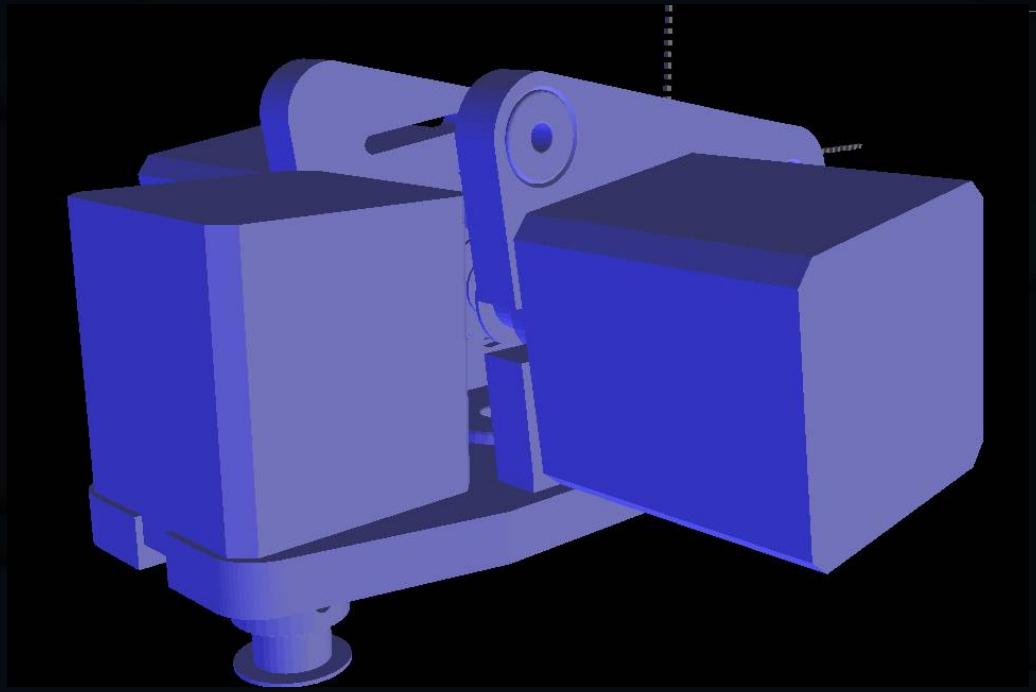
Making the model: Sphere/Cake Wedge

<<NameOfObject>> = **Sphere**(<<xCenter>>,<<yCenter>>,<<zCenter>>)

<<NameOfObject>> = **ArcX**(<<x1>>,<<x2>>,
<<radius1>>,<<radius2>>,
<<angle1>>,<<angle2>>,
<<steps>>)



Making the model: Full .stl Object

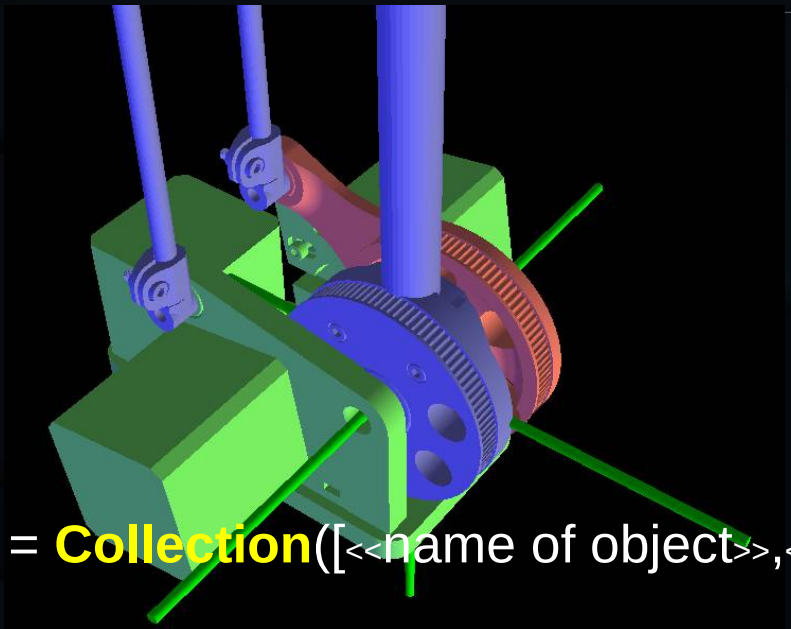


<<NameOfObject>> = **AsciiSTL**(<<full filename including path of .stl file>>)

Seriously Recommended

Set the origin at a rotation axis

Making the Model: Collections



<<NameOfCollection>> = **Collection**([<<name of object>>,<<name of object>>,....])

<<NameOfCollection>> = **Collection**([<<name of collection>>,....])

Also possible (and widely used) to add an object to an existing collection

e.g. myCollection = Collection([myCollection , box])

Altering the Model: Move/Rotate/Scale

<<NameOfCollection>> = **Translate**(**[**<<name of collection>>**]**,<<x>>,<<y>>,<<z>>)

<<NameOfCollection>> = **Rotate**(**[**<<name of collection>>**]**,<<angle>>,<<x>>,<<y>>,<<z>>)

<<NameOfCollection>> = **Scale**(**[**<<name of collection>>**]**,<<%x>>,<<%y>>,<<%z>>)

(% is a decimal e.g. 0.10 = 10%)

Animating the Model: Move/Rotate

```
<<NameOfCollection>> = HalTranslate(
    [<<name of collection>>],
    <<halComponentName>>,
    ”jointName”,
    <<xdistance>>,<<ydistance>>,<<zdistance>>)

<<NameOfCollection>> = HalRotate(
    [<<name of collection>>],
    <<halComponentName>>,
    ”jointName”,
    <<angle>>,
    <<xaxis>>,<<yaxis>>,<<zaxis>>)
```

Where we indicate the axis with 1 or -1 or 0

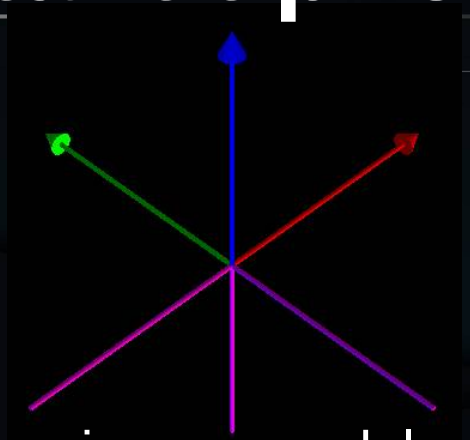
Tips about Rotation

- A model made with cylinders / boxes / spheres etc has to be **moved around** as you create its parts, so that the current rotation axis is always at the current origin
- .STL model parts are added to the scene in the same pose as they were created i.e **ideally with a rotation point at their origin**
- **Rotation is always around an axis through the current origin**
- The origin does not move – Translate and Rotate the **model** as you add parts, relative to a **STATIONARY origin**

More Important Tips

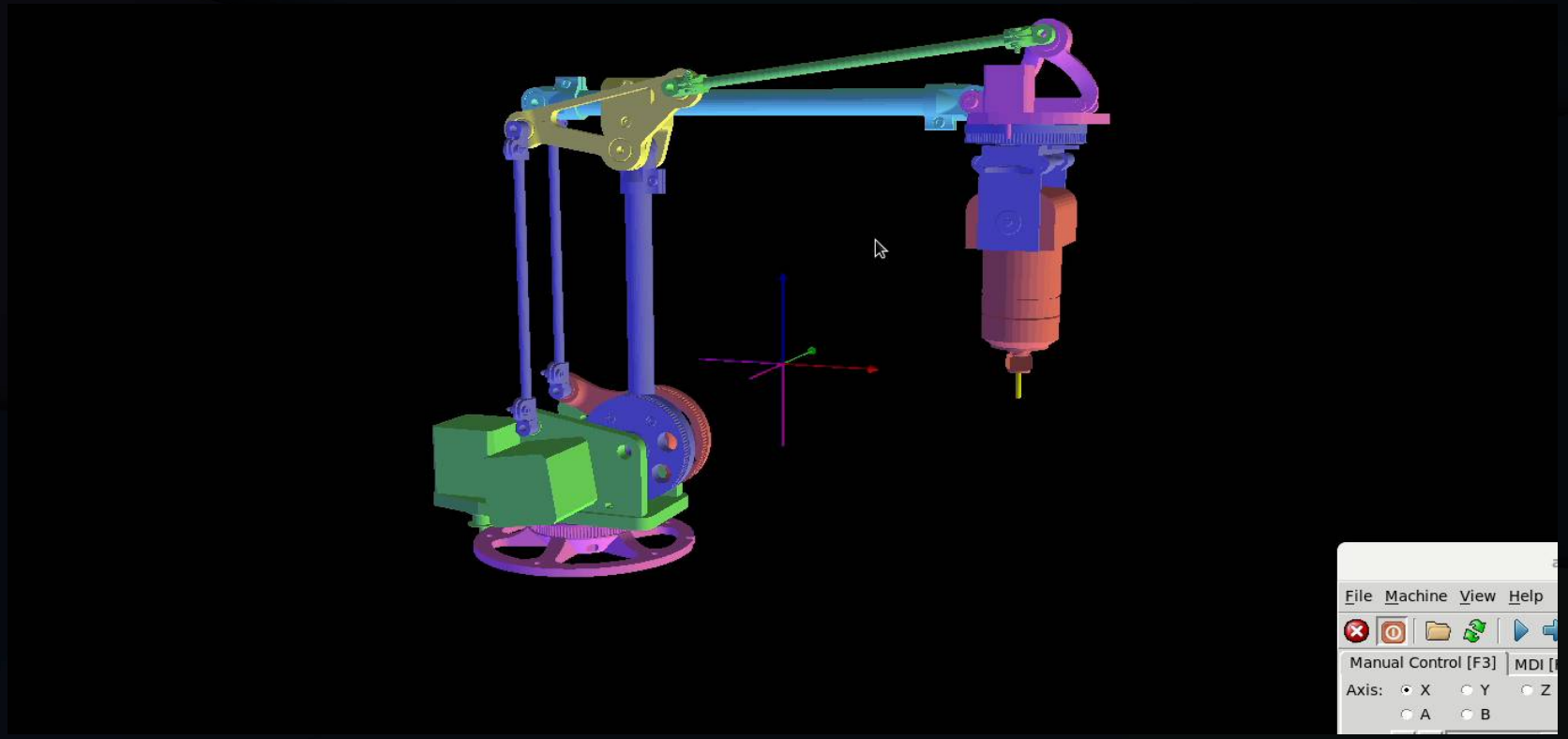
- The **whole chain (collection)** is translated/rotated by a HalRotate/HalTranslate so start assembly at the cutting tip and work backwards, building a collection step by step
- Then fill in the animation with HalTranslate and HalRotate’s at the appropriate points, ending back at the cutting tip
- Run the simulation after each addition, to check

Most important tip for beginners



- Create axes as a collection in your model which is always at the origin
- Add axes collection unmoved to your “model” collection
- While you add each element in your chain, you will always be able to see where your **current origin** is located
- Make a comment at each point in your script as to where the current origin is
- **Ignore this at your peril !!!!**

Example: Visualisation of 5D Robot



Writing the Vismach Script

Name of Vismach script

```
halComponent = hal.component("CNCClubVismach")
halComponent.newpin("jointX", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.newpin("jointY", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.newpin("jointZ", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.ready()
```

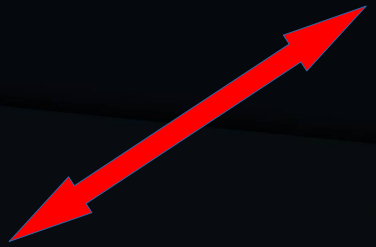
Your Chosen Names

Any name – but best called halComponent

“Minimum” Vismach Script components

```
#!/usr/bin/python2
from vismach import *
import hal
import math
import sys
halComponent = hal.component("CNCClubVismach")
halComponent.newpin("jointX", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.newpin("jointY", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.newpin("jointZ", hal.HAL_FLOAT, hal.HAL_IN)
halComponent.ready()
tooltip = Capture()
.......
work = Capture()
.......

main(model , tooltip , work , 1000)
```

Window View Size

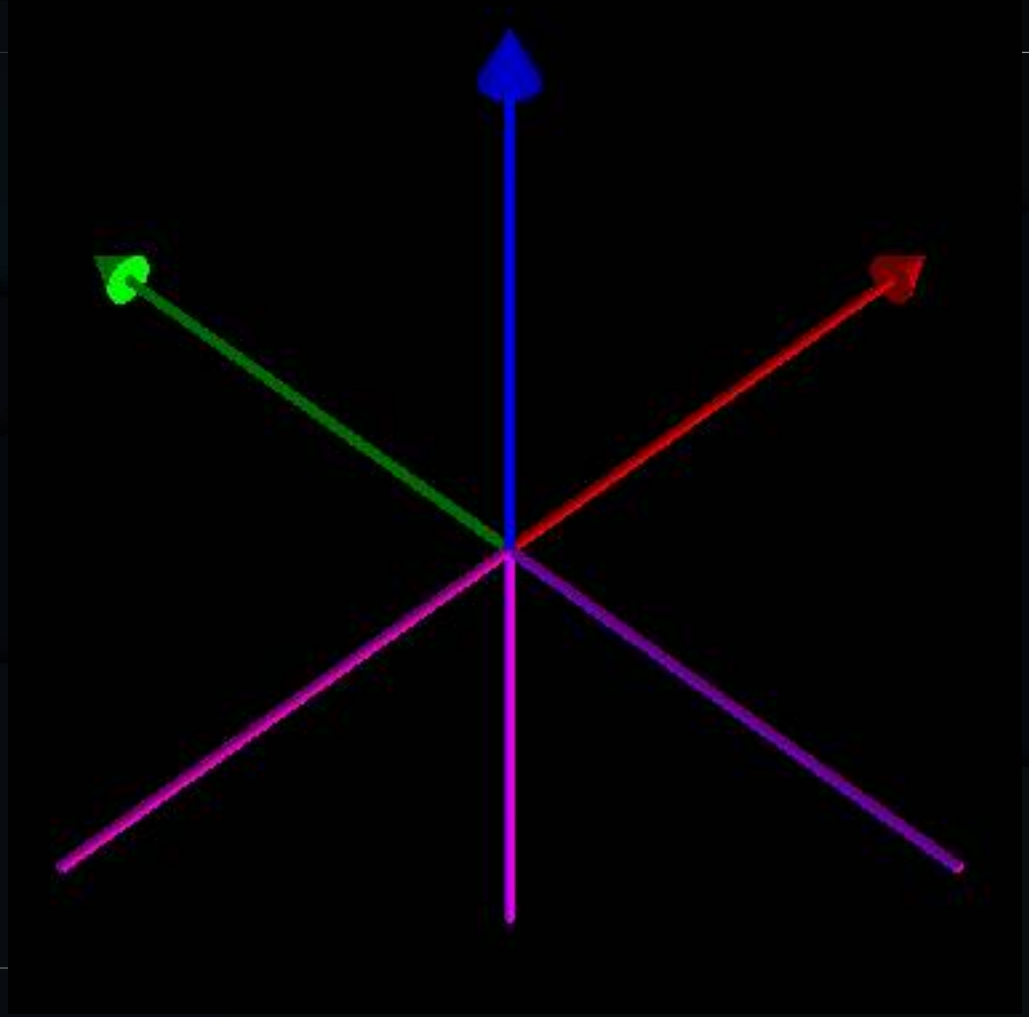


First step: Give a shape to the tool

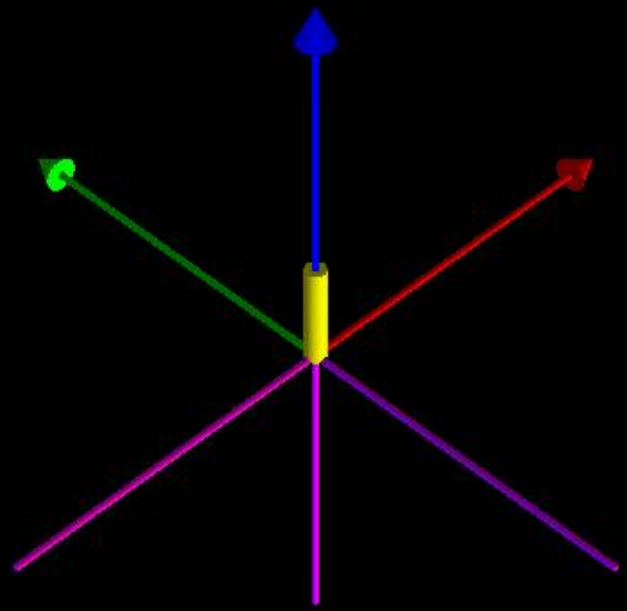
```
tooltip = Capture()

toolshape = CylinderZ(30,10,0,10)
toolshape = Color([1,1,0,1],[toolshape])
Aassembly = Collection([
    toolshape,
    tooltip
])

work = Capture()

model = Collection([Aassembly , work])

main(model , tooltip , work , 1000)
```

Second Step: Add some Axes around the Origin

```
....
X=CylinderX(-100,1,100,1)
Y=CylinderY(-100,1,100,1)
Z=CylinderZ(-100,1,100,1)
axes = Collection([X,Y,Z])
axes = Color([0,1,0,1],[axes])

work = Capture()

model = Collection( [
    Aassembly,
    axes,
    work
] )
```

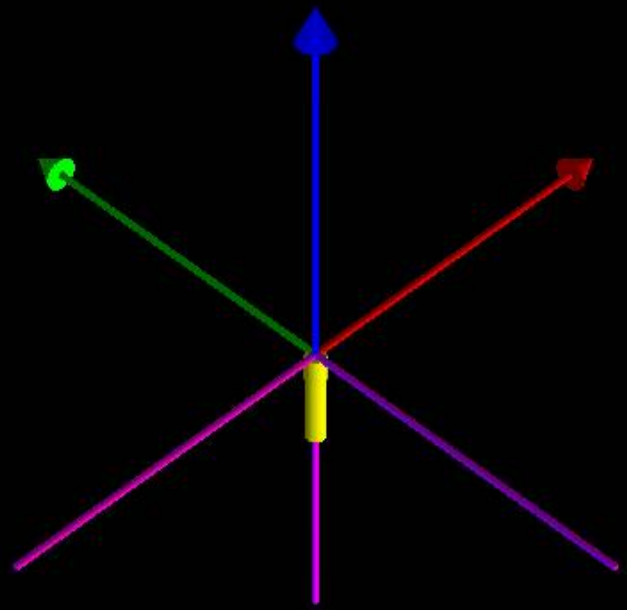


Tool is now visible at the Origin

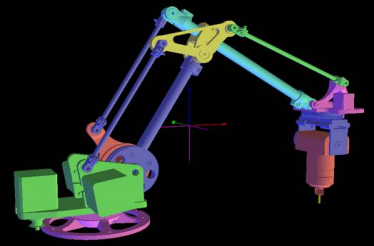


(But not much else)

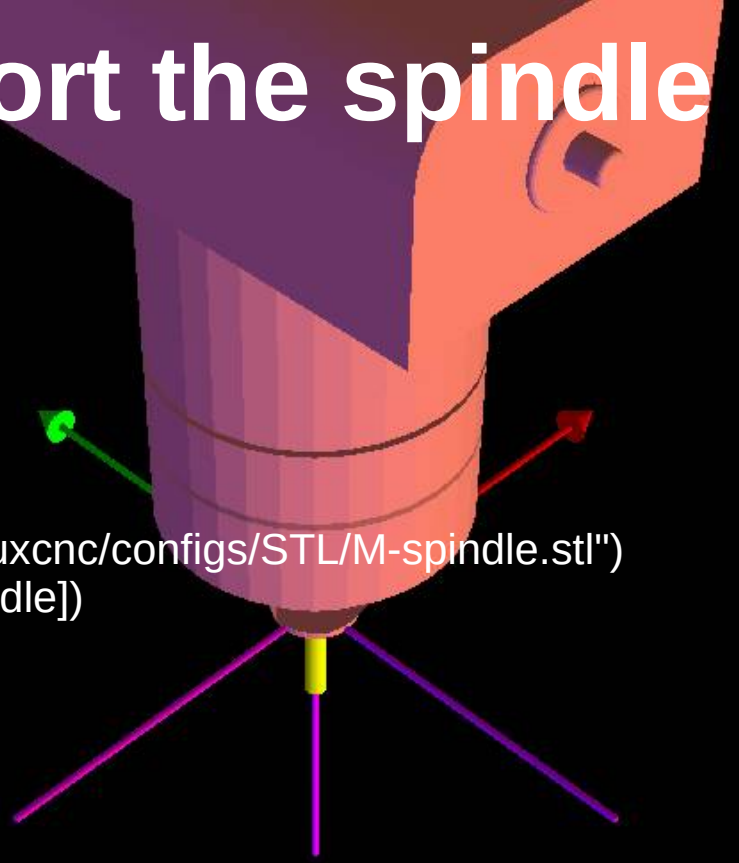
Shift Tool down so origin is where Spindle will be added



Aassembly = Translate([Aassembly],0,0,-30)

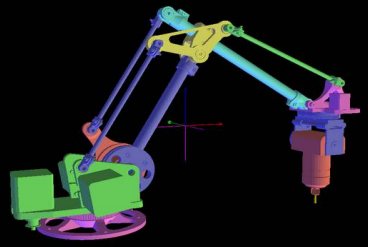


Import the spindle



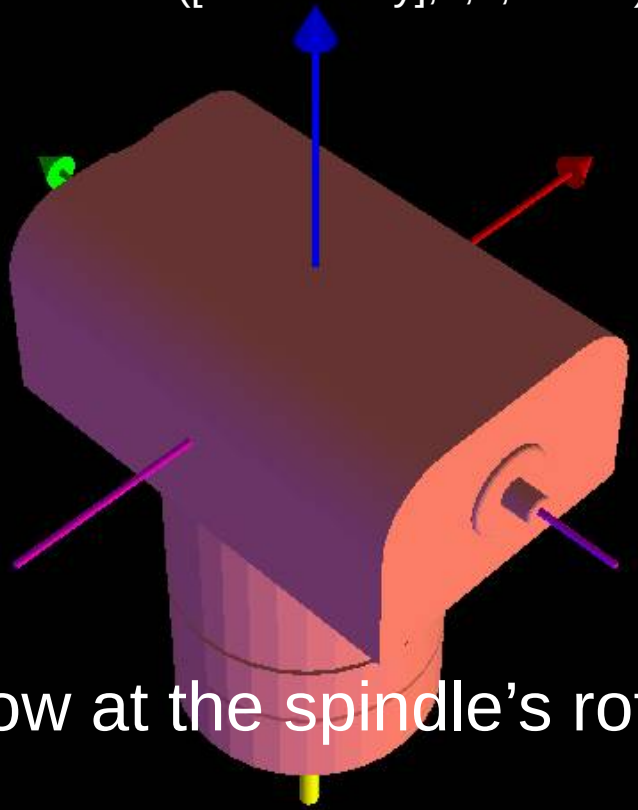
```
spindle = AsciiSTL("/home/cecil/linuxcnc/configs/STL/M-spindle.stl")
spindle = Color([1.0,0.5,0.5,1],[spindle])

Aassembly = Collection([
    Aassembly,
    spindle
])
```

Tool is now in the spindle collet



Translate the Assembly down

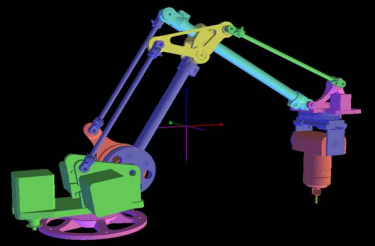
Aassembly = Translate([Aassembly],0,3,-171.7)

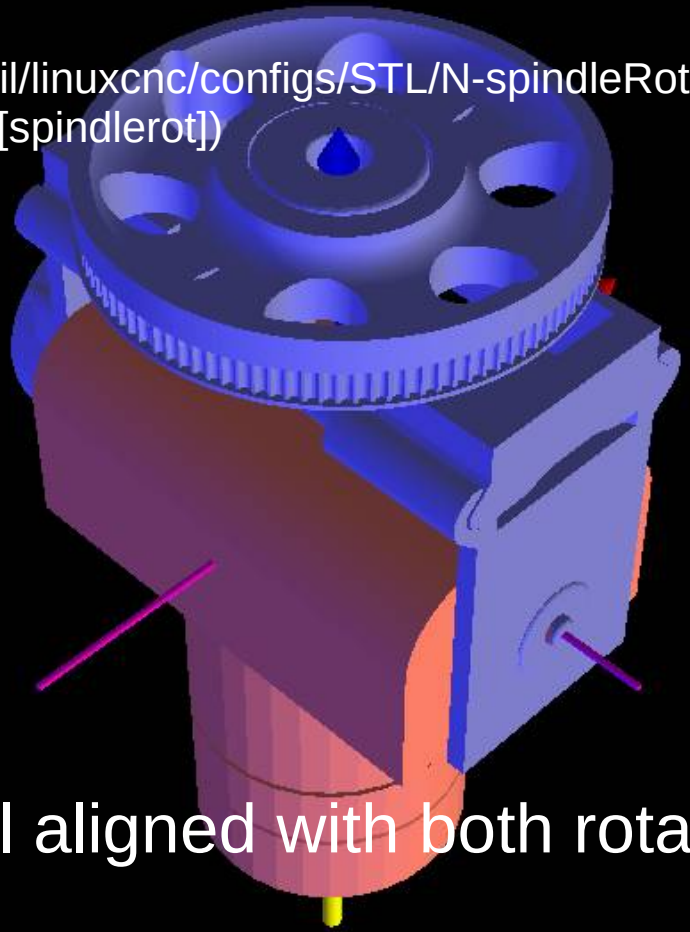


Origin is now at the spindle’s rotation axis

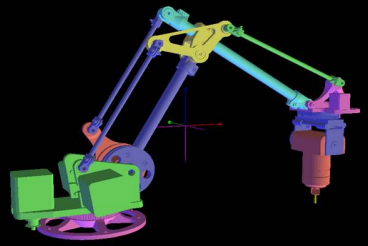
Import

```
spindlerot = AsciiSTL("/home/cecil/linuxcnc/configs/STL/N-spindleRot.stl")
spindlerot = Color([0.5,0.5,1.0,1],[spindlerot])

Aassembly = Collection([
    Aassembly,
    spindlerot
])
```



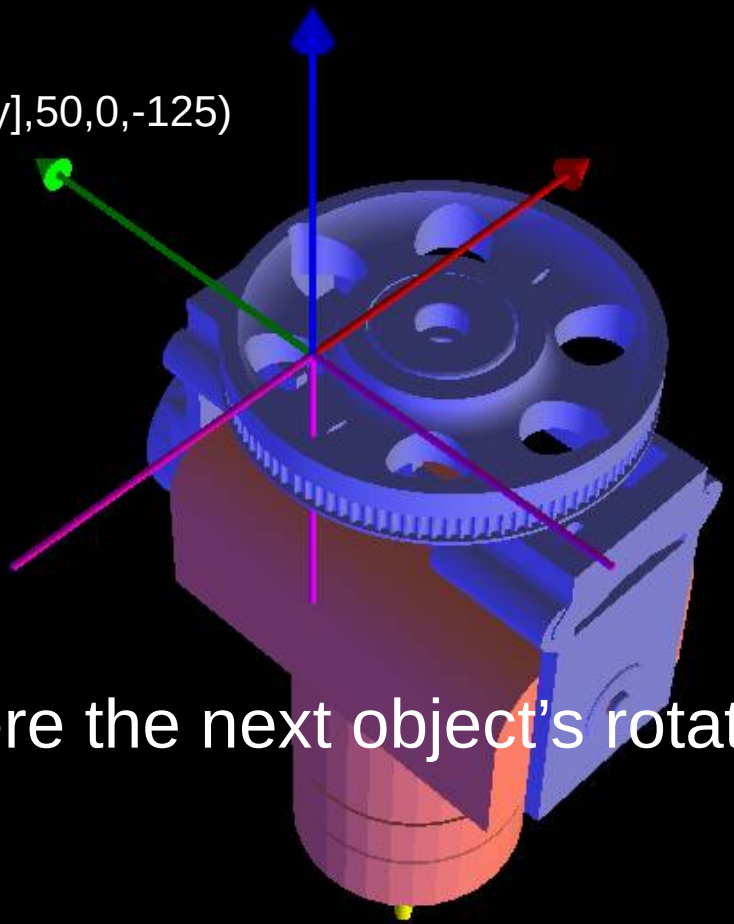


Origin is still aligned with both rotation axes



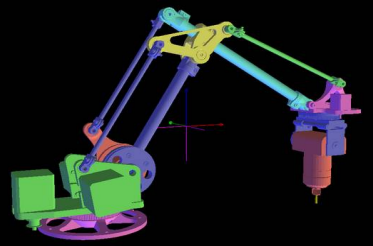
Translate

Aassembly = Translate([Aassembly],50,0,-125)

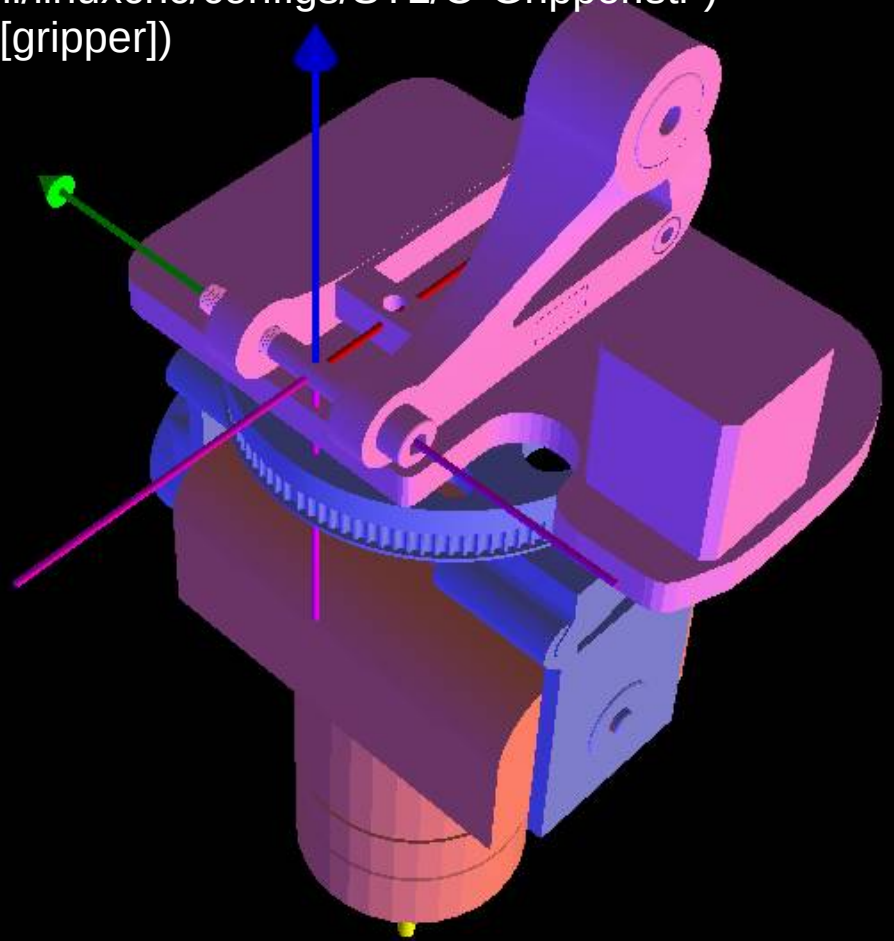


Origin is now where the next object’s rotation axis will be

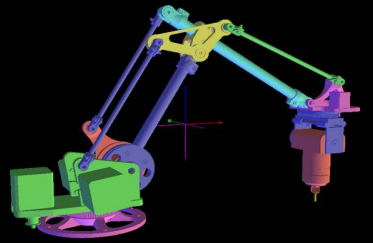
Import



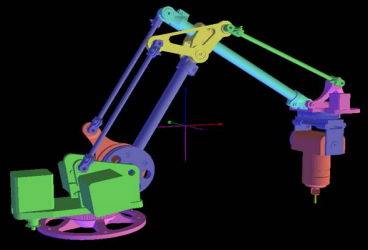
```
gripper = AsciiSTL("/home/cecil/linuxcnc/configs/STL/O-Gripper.stl")
gripper = Color([1.0,0.5,1.0,1],[gripper])

Aassembly = Collection([
    Aassembly,
    gripper
])
```



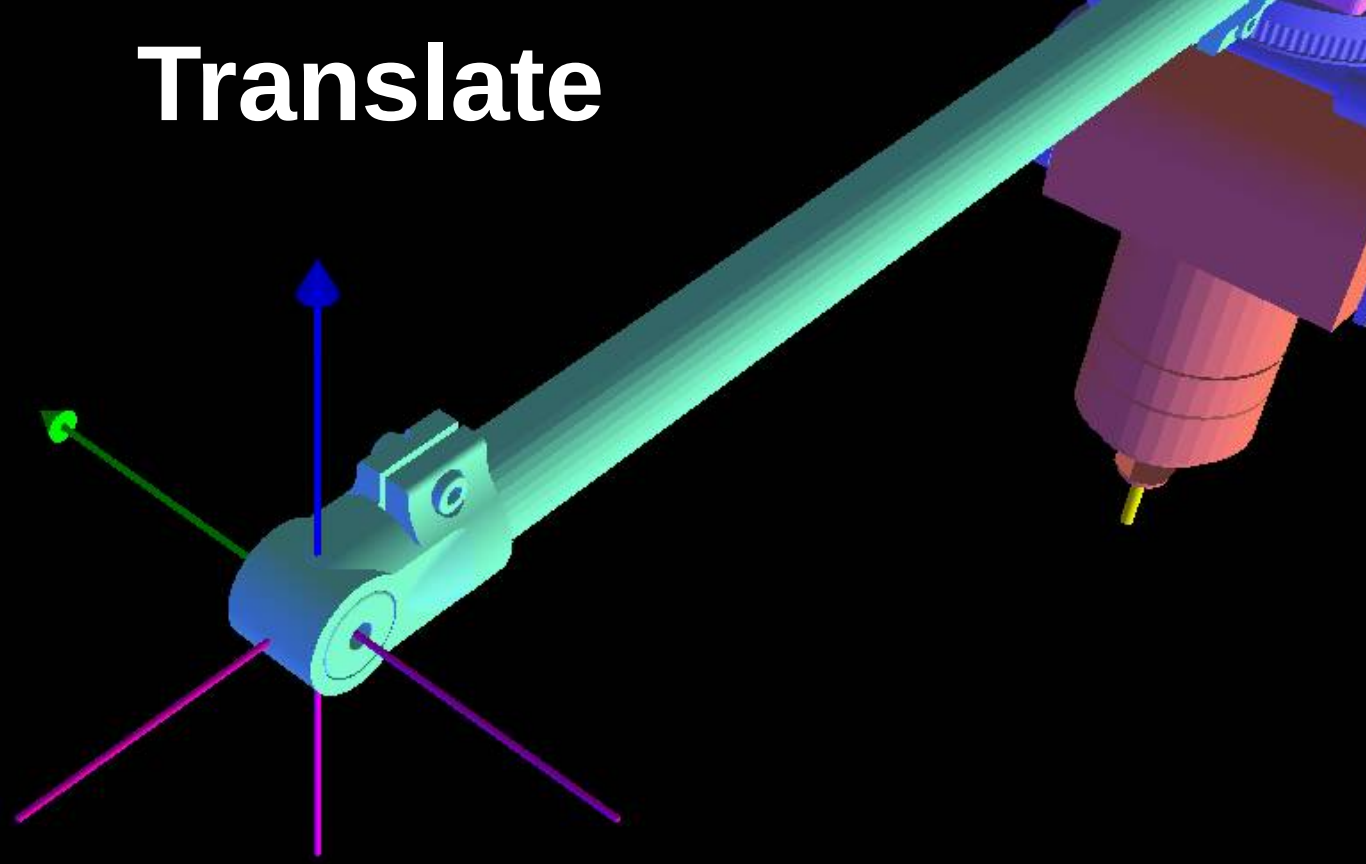
Import



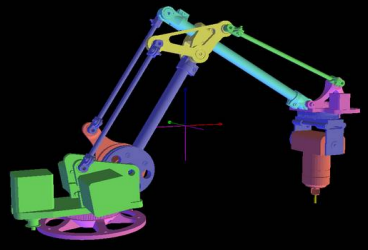
```
bigarm = AsciiSTL("/home/cecil/linuxcnc/configs/STL/H-BigArm.stl")
bigarm = Color([0.5,1.0,1.0,1],[bigarm])

Aassembly = Collection([
    Aassembly,
    bigarm
])
```



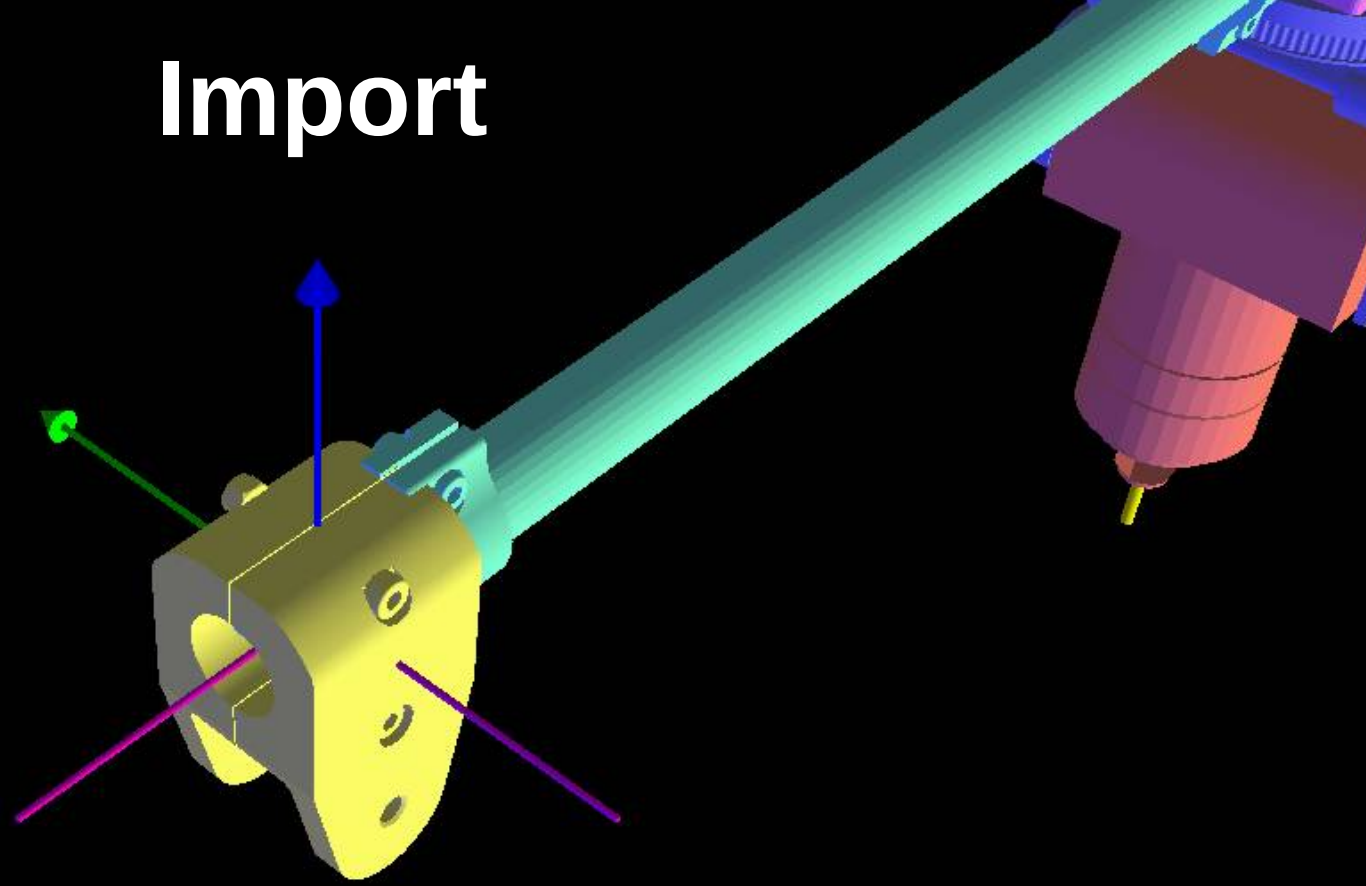
Translate



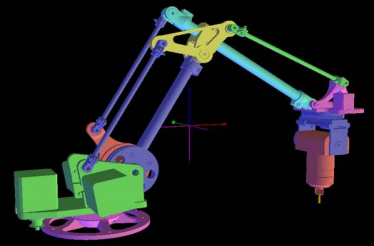
Aassembly = Translate([Aassembly],500,0,-1)



Import

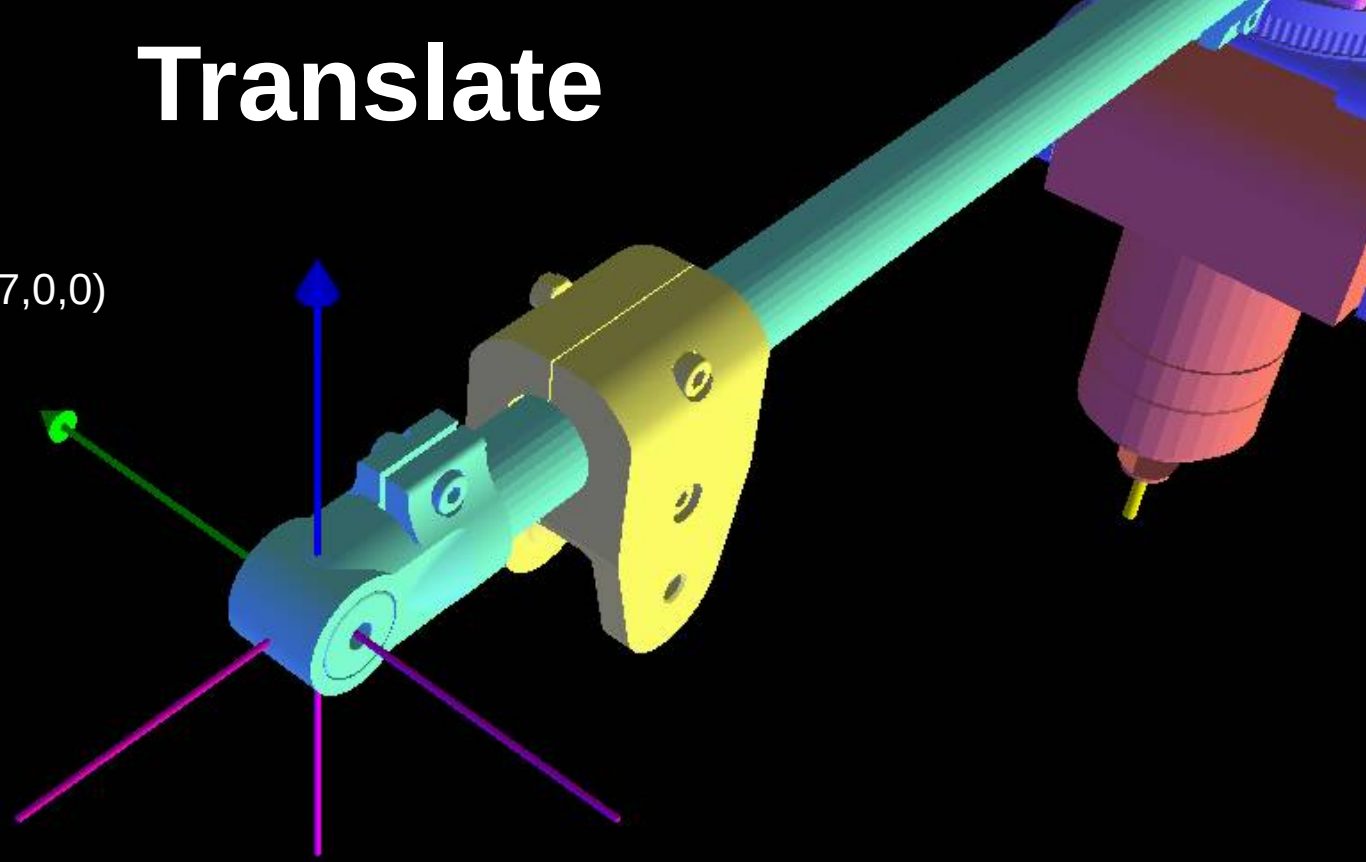


```
bracket = AsciiSTL("/home/cecil/linuxcnc/configs/STL/I-Bracket.stl")
bracket = Color([1.0,1.0,0.5,1],[bracket])
```

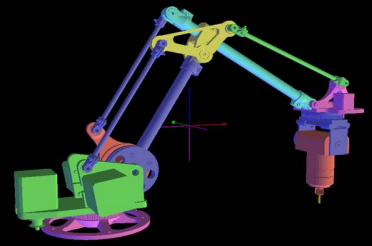


Translate

```
bracket = Translate([bracket],117,0,0)

Aassembly = Collection([
    Aassembly,
    bracket
])
```

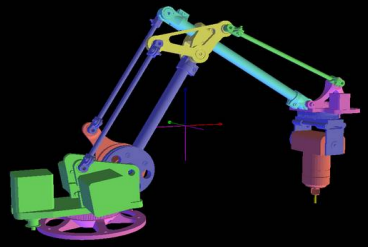


Translate the object **first**, then combine it into the collection

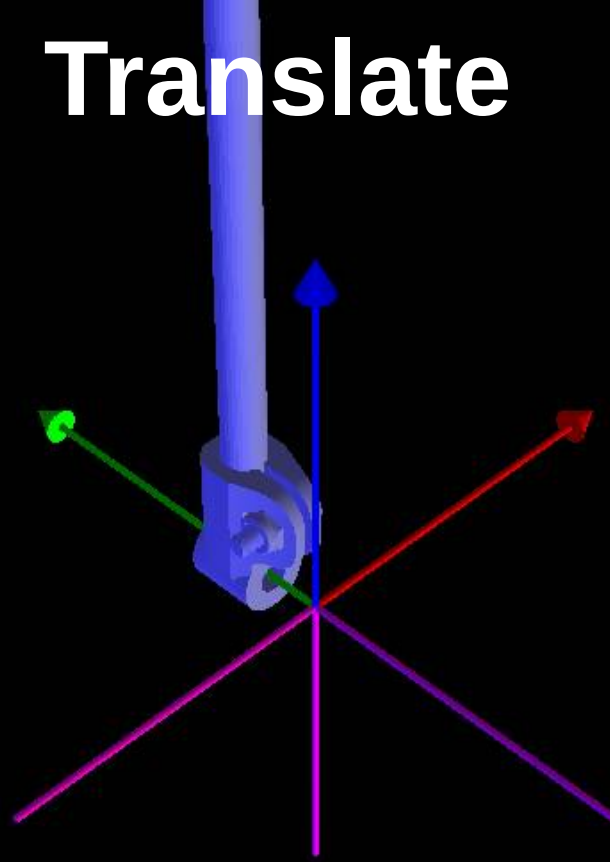


Import

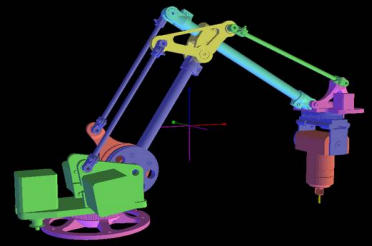
```
arm180trans = AsciiSTL("/home/cecil/linuxcnc/configs/STL/G-arm180trans.stl")
arm180trans = Color([0.5,0.5,1.0,1],[arm180trans])
arm180trans = Translate([arm180trans],0,16,0)

Aassembly = Collection([
    Aassembly,
    arm180trans
])
```



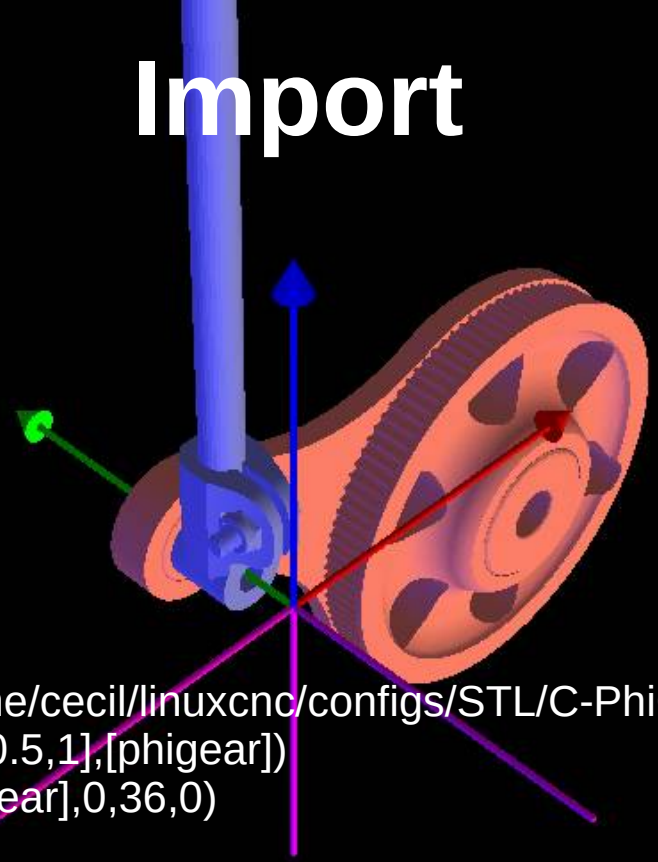
Translate



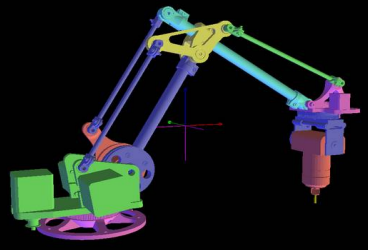
Aassembly = Translate([Aassembly],0,0,360)



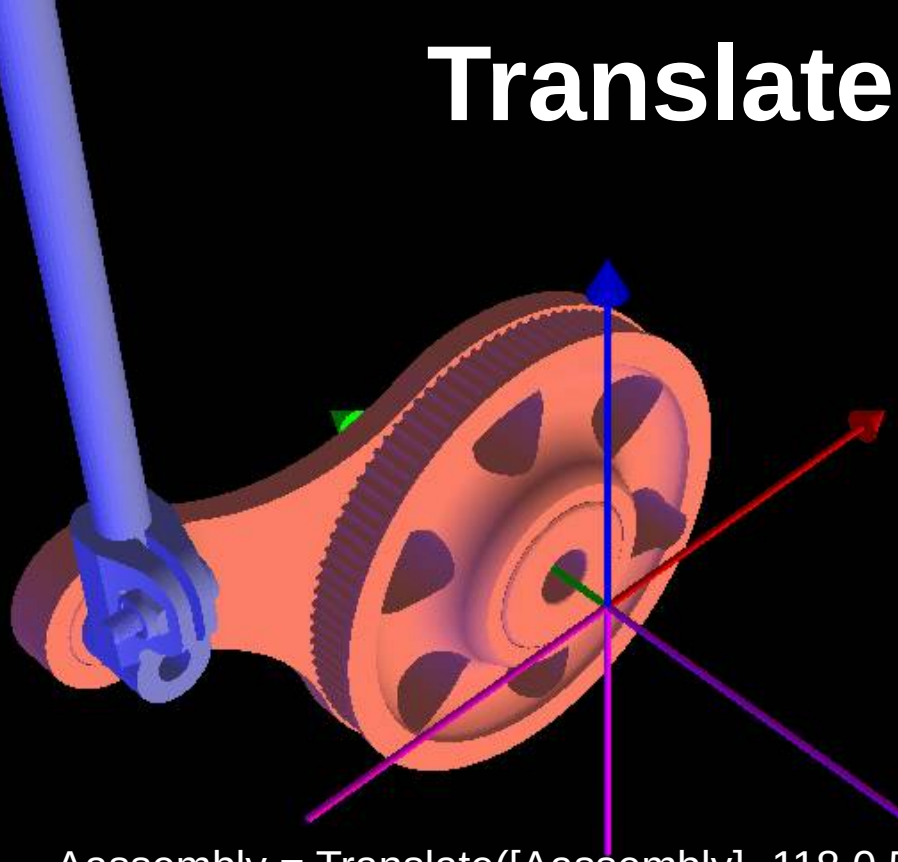
Import



```
phigear = AsciiSTL("/home/cecil/linuxcnc/configs/STL/C-PhiGear.stl")
phigear = Color([1.0,0.5,0.5,1],[phigear])
phigear = Translate([phigear],0,36,0)

Aassembly = Collection([
    Aassembly,
    phigear
])
```



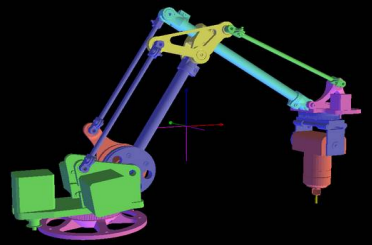
Translate



Aassembly = Translate([Aassembly],-118,0,55)

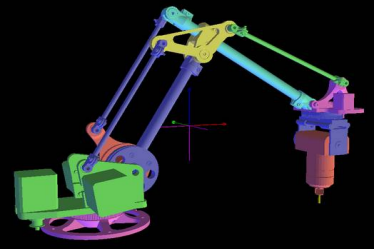
We are now at the final rotation axis of this collection

Start a new Collection

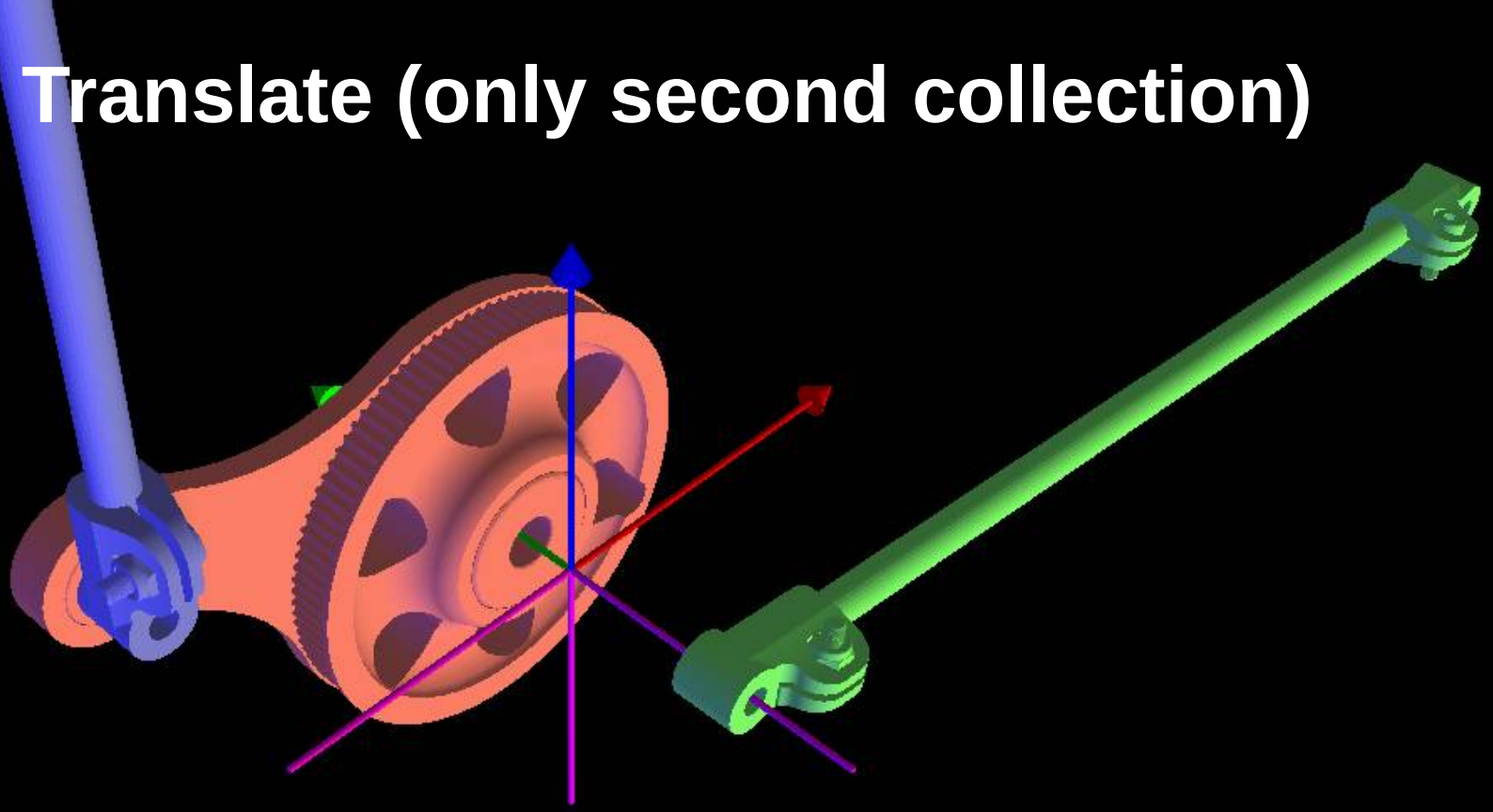


```
Bassembly = AsciiSTL("/home/cecil/linuxcnc/configs/STL/E-arm190.stl")
Bassembly = Color([0.5,1.0,0.5,1],[Bassembly])
```

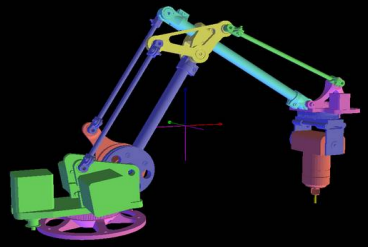
Note that we leave the first collection where it is (with its last axis at the origin). We now **only** move the second collection as we assemble it.



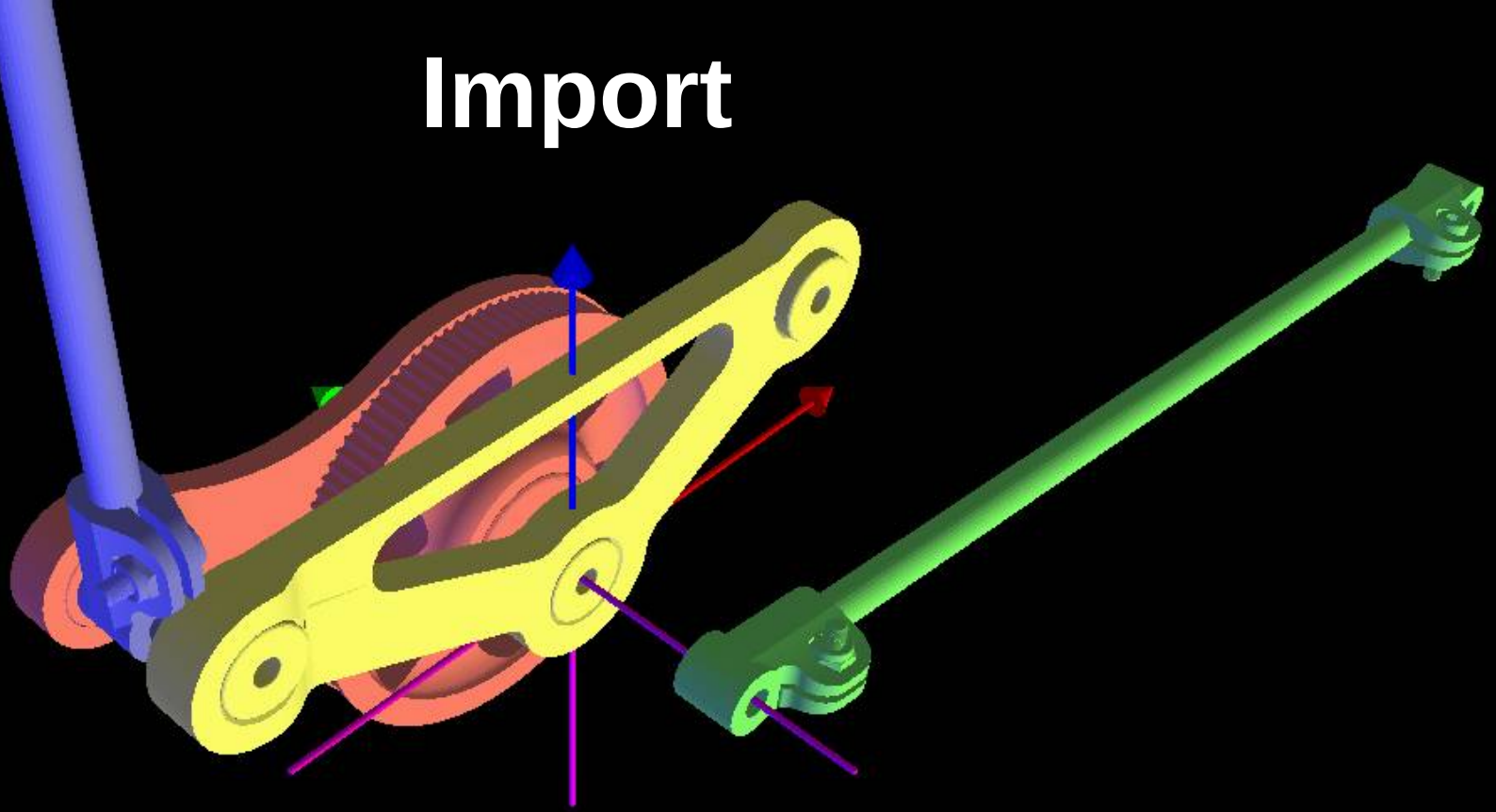
Translate (only second collection)



Bassembly = Translate([Bassembly],390,-57,0)
Bassembly = Rotate([Bassembly],6.5,0,1,0)



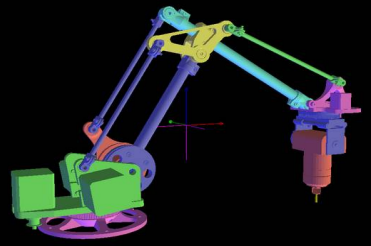
Import



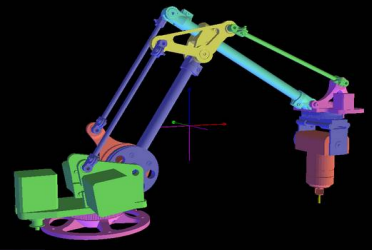
```
triangle = AsciiSTL("/home/cecil/linuxcnc/configs/STL/F-Triangle.stl")
triangle = Color([1.0,1.0,0.5,1],[triangle])
```

Translate

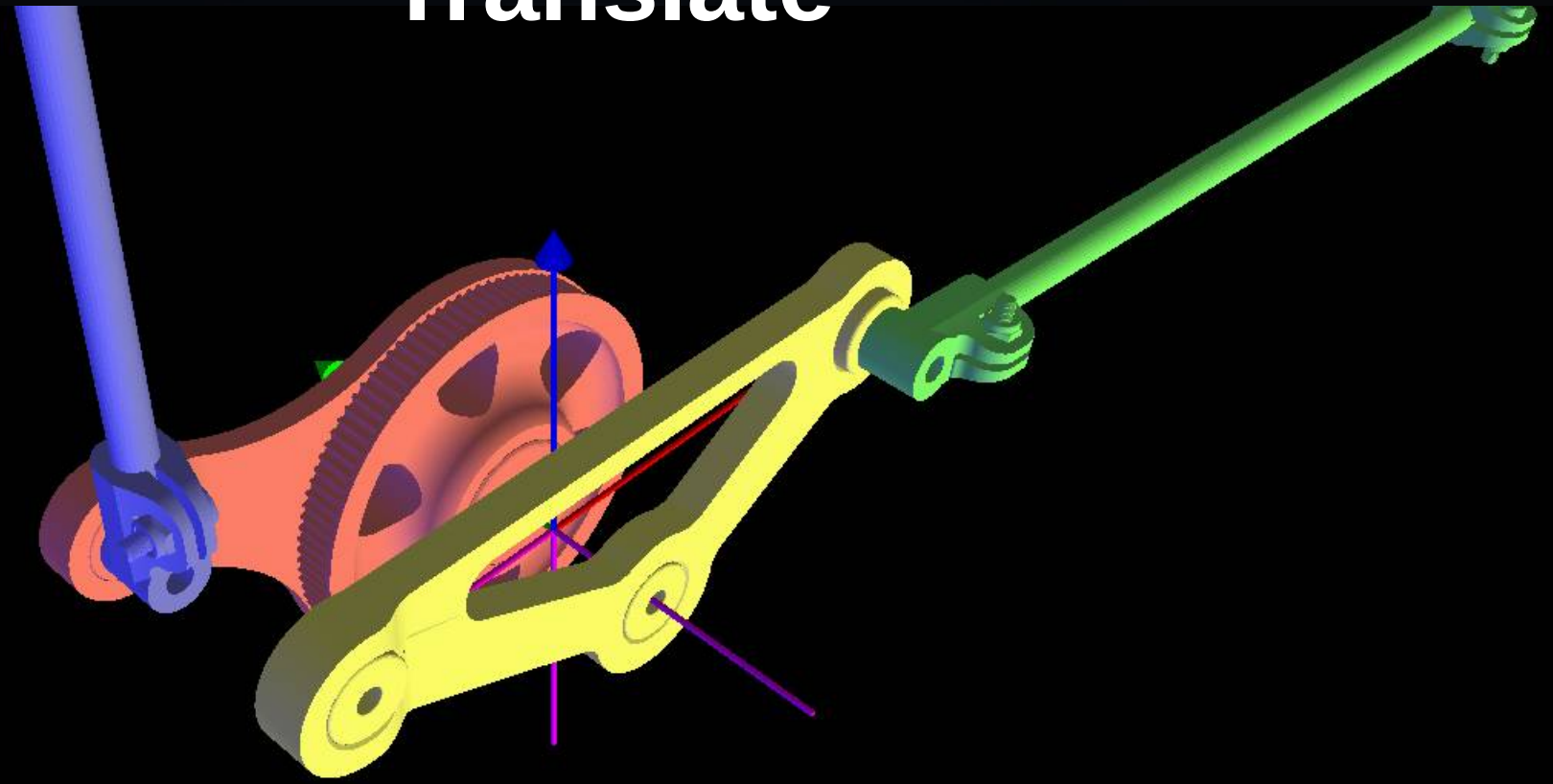
```
triangle = Translate([triangle],-86,-36,-51.5)

Bassembly = Collection([
    Bassembly,
    triangle
])
```



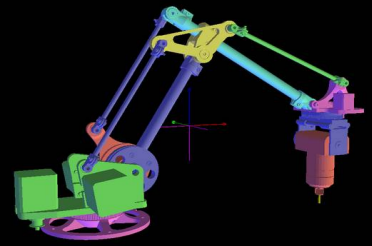
This time, the origin of the object was not in the correct place.
So we translate the object before we add it to the collection.



Translate



Bassembly = Translate([Bassembly],86,0,51.5)

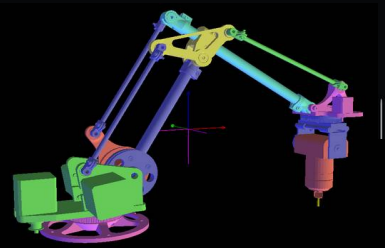


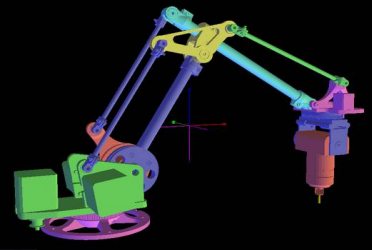
Rotate

Bassembly = Rotate([Bassembly],-14.0,0,1,0)

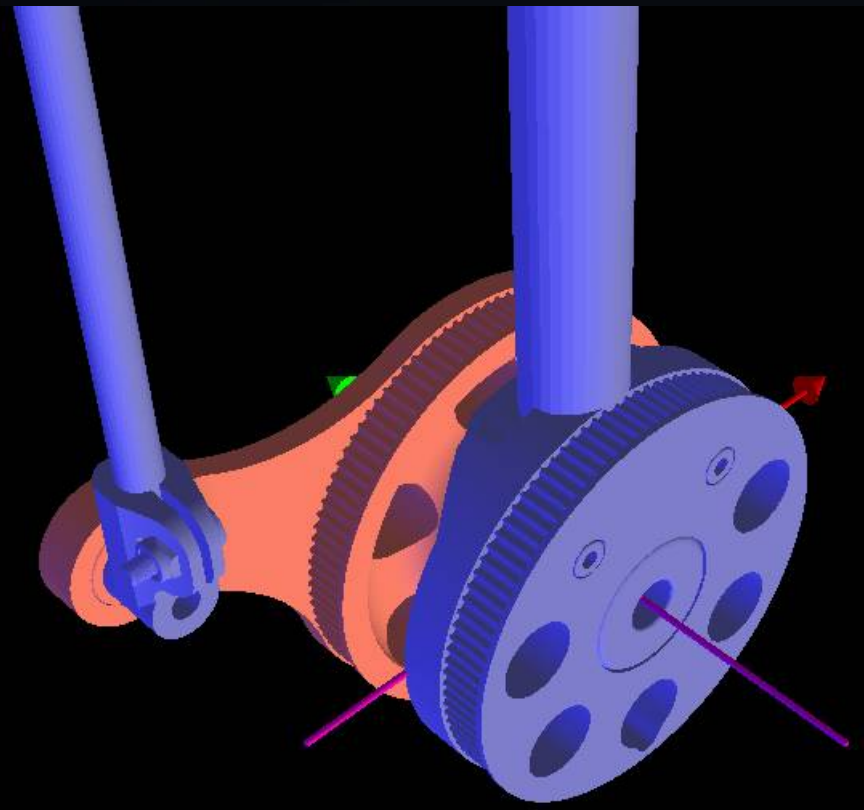
Import

```
thetagear = AsciiSTL("/home/cecil/linuxcnc/configs/STL/D-ThetaGear.stl")
thetagear = Color([0.5,0.5,1.0,1],[thetagear])

Bassembly = Collection([
    Bassembly,
    thetagear
])
```

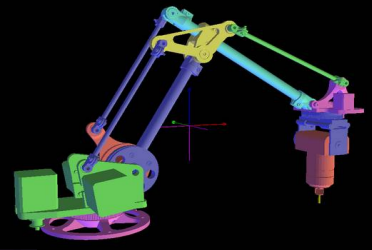




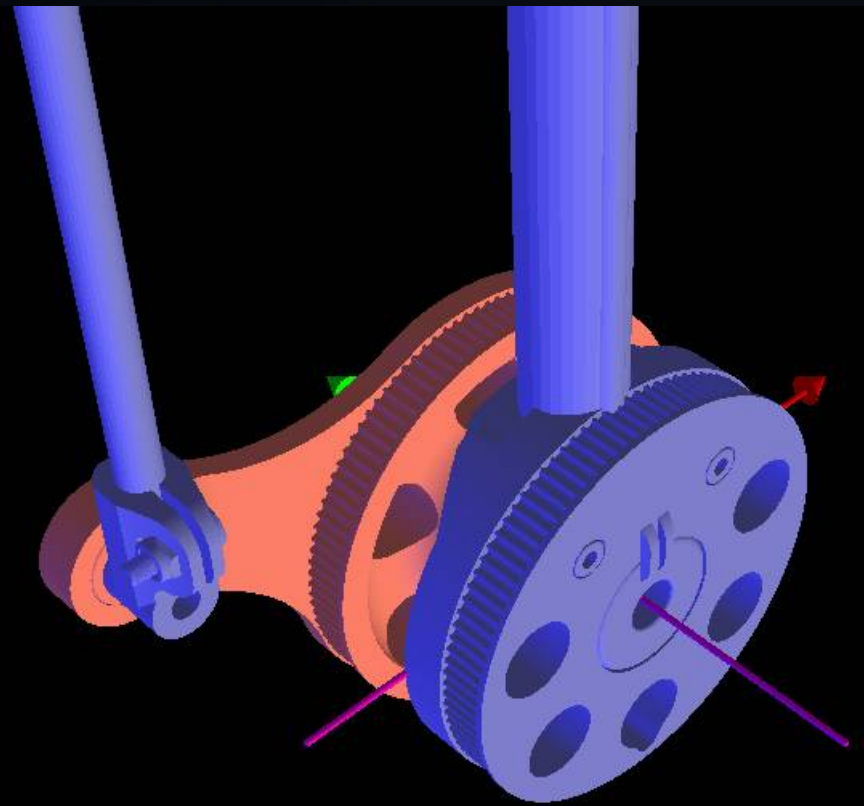
Translate



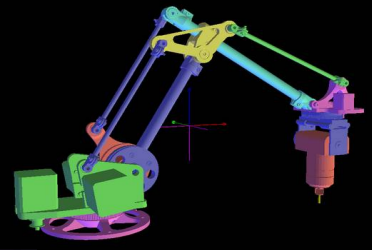
Bassembly = Translate([Bassembly],0,0,360)



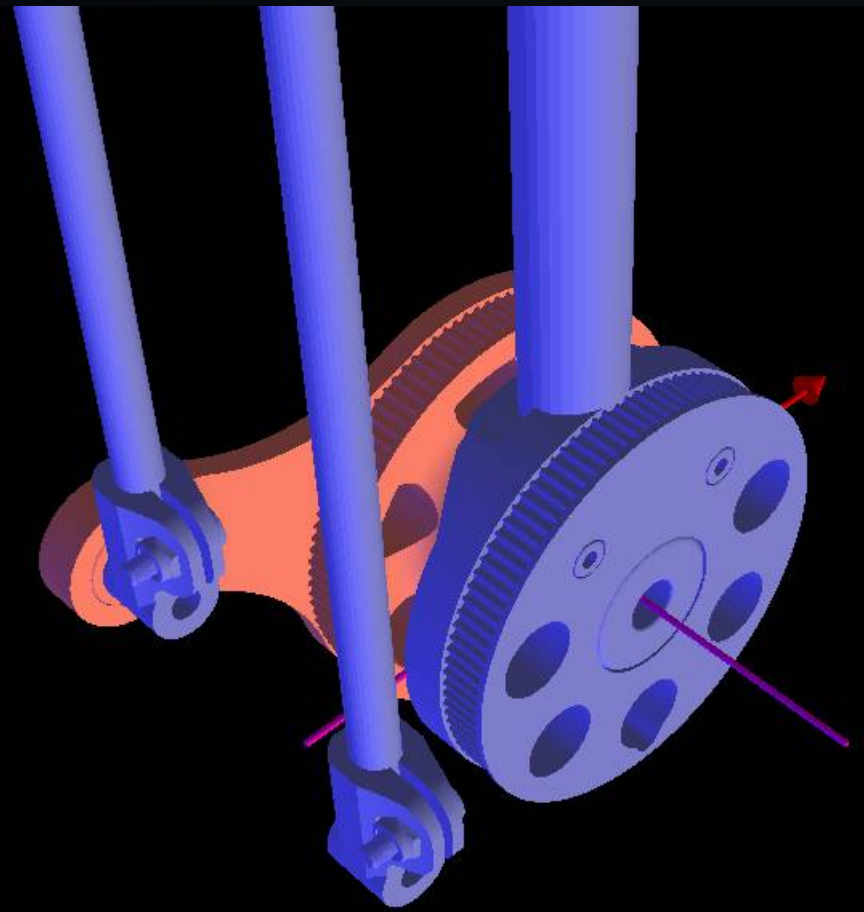
Import



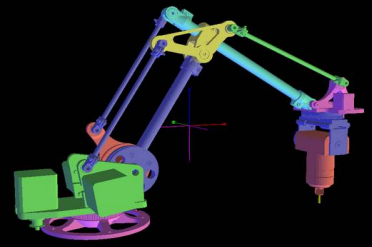
```
arm180cis = AsciiSTL("/home/cecil/linuxcnc/configs/STL/K-arm180cis.stl")
arm180cis = Color([0.5,0.5,1.0,1],[arm180cis])
```



Translate



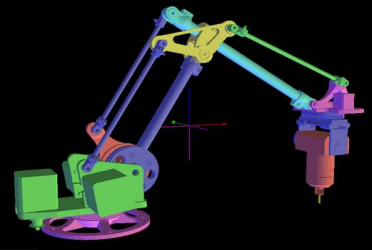
arm180cis = Translate([arm180cis],-118,-45,24)



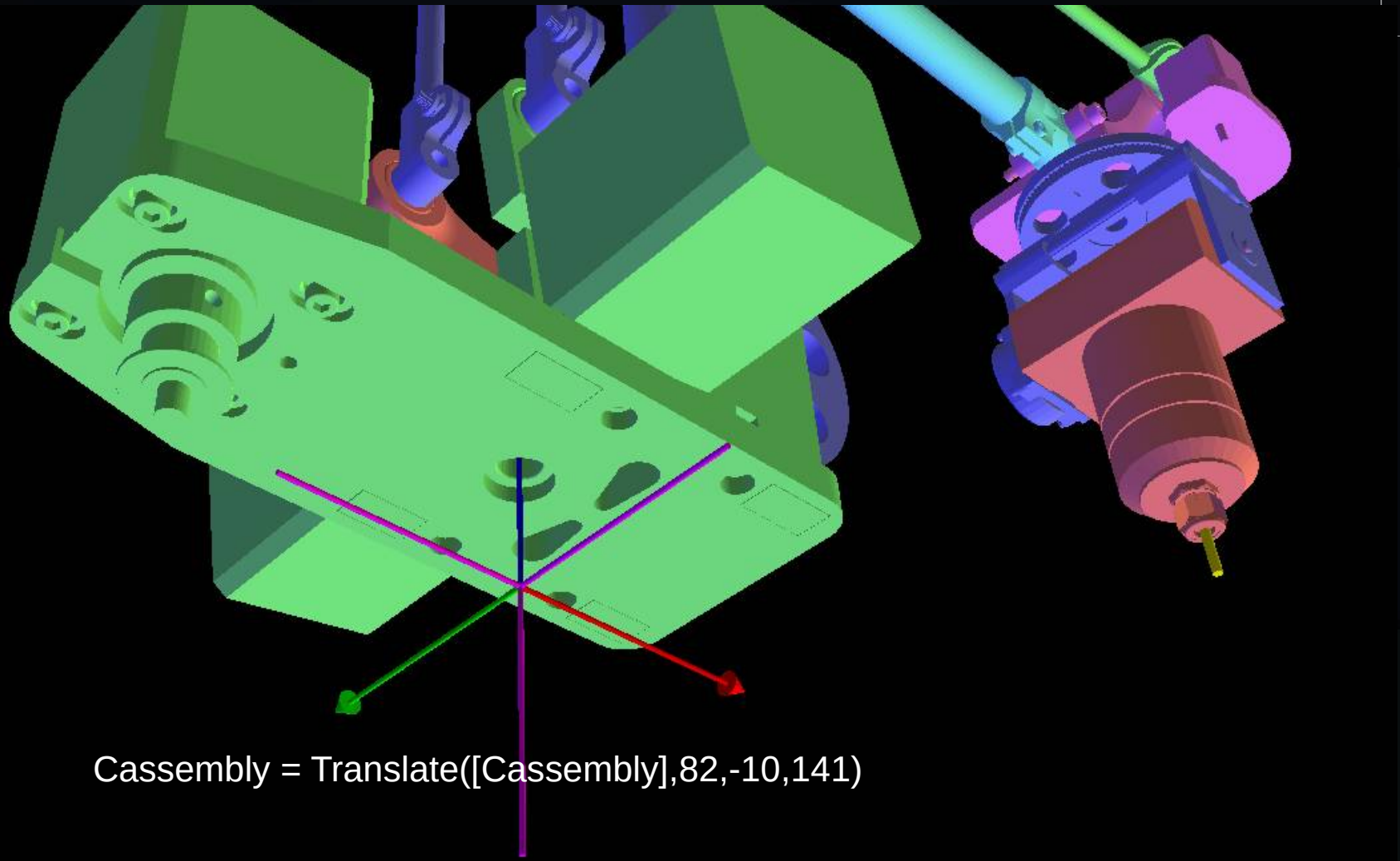
Import

```
mechanism = AsciiSTL("/home/cecil/linuxcnc/configs/STL/B-Mechanism.stl")
mechanism = Color([0.5,1.0,0.5,1],[mechanism])
```

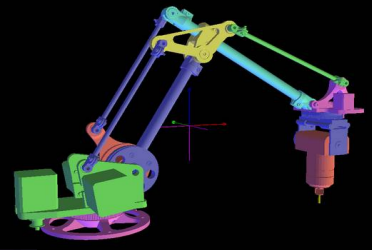
```
Cassembly = Collection([
    mechanism,
    arm180cis,
    Bassembly,
    Aassembly
])
```



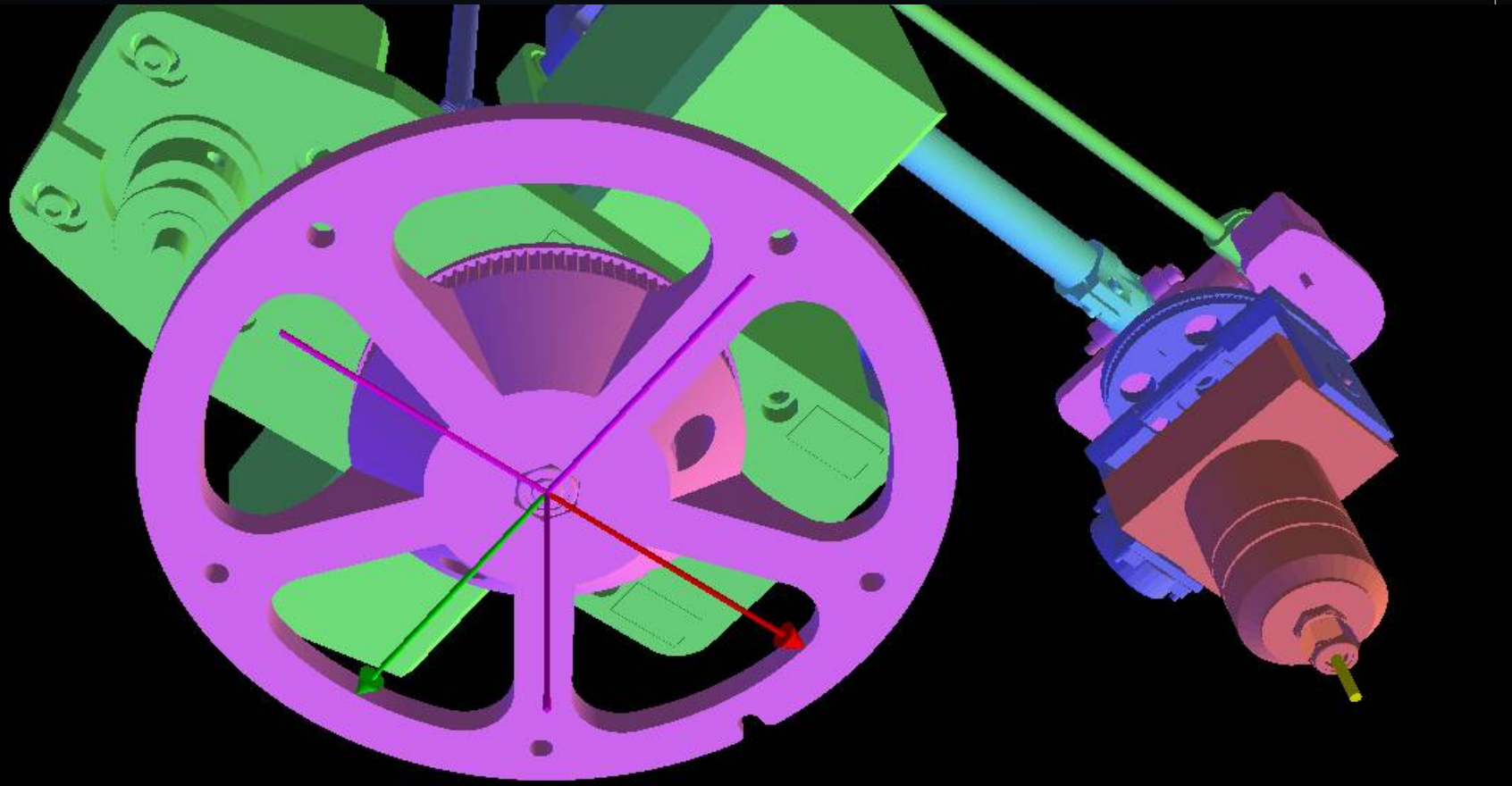
Translate



Cassembly = Translate([Cassembly],82,-10,141)



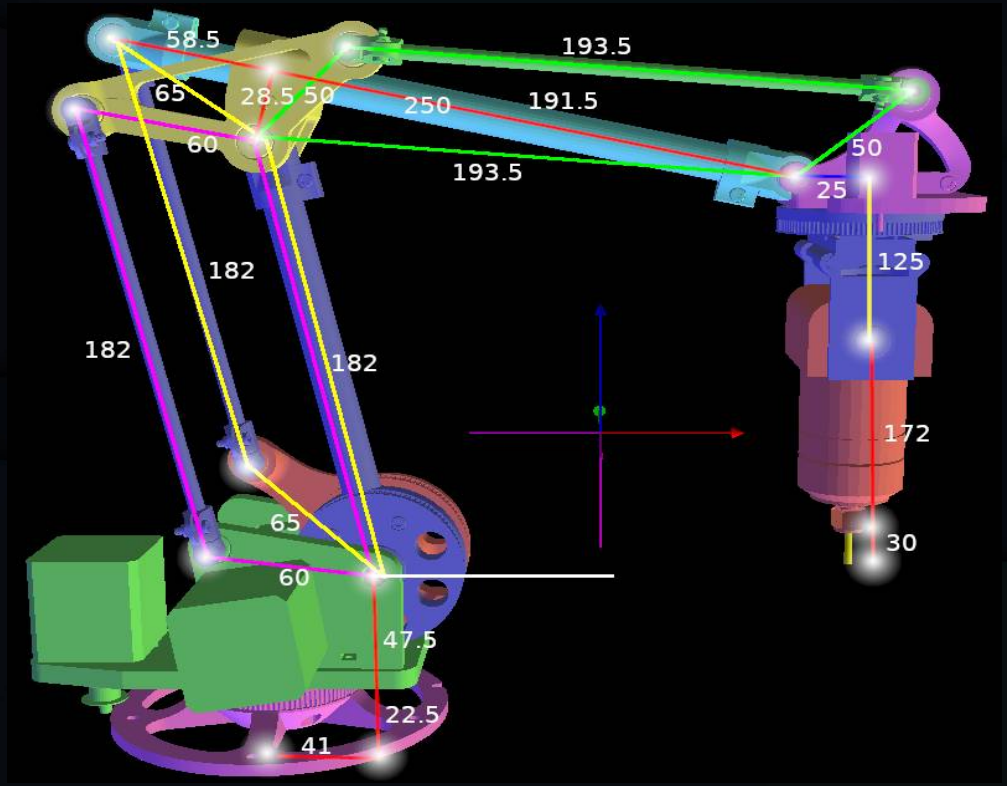
Import



```
base = AsciiSTL("/home/cecil/linuxcnc/configs/STL/L-Base.stl")
base = Color([1.0,0.5,1.0,1], [base])
```

Static Model Complete

```
Cassembly = Translate([Cassembly],-250,0,-250)
base = Translate([base],-250,0,-250)

work = Capture()


model = Collection([
    axes,
    Cassembly,
    base,
    work
])
```



Animating the Model: Move/Rotate

```
<<NameOfCollection>> = HalTranslate(
    [<<name of collection>>],
    <<halComponentName>>,
    ”jointName”,
    <<xdistance>>,<<ydistance>>,<<zdistance>>)

<<NameOfCollection>> = HalRotate(
    [<<name of collection>>],
    <<halComponentName>>,
    ”jointName”,
    <<angle>>,
    <<xaxis>>,<<yaxis>>,<<zaxis>>)
```

Where we indicate the axis with 1 or -1 or 0