Camotics

Rotary camotics

Rotary Camotics

Rotary Toolpaths with Meshlab, Pycam and Camotics

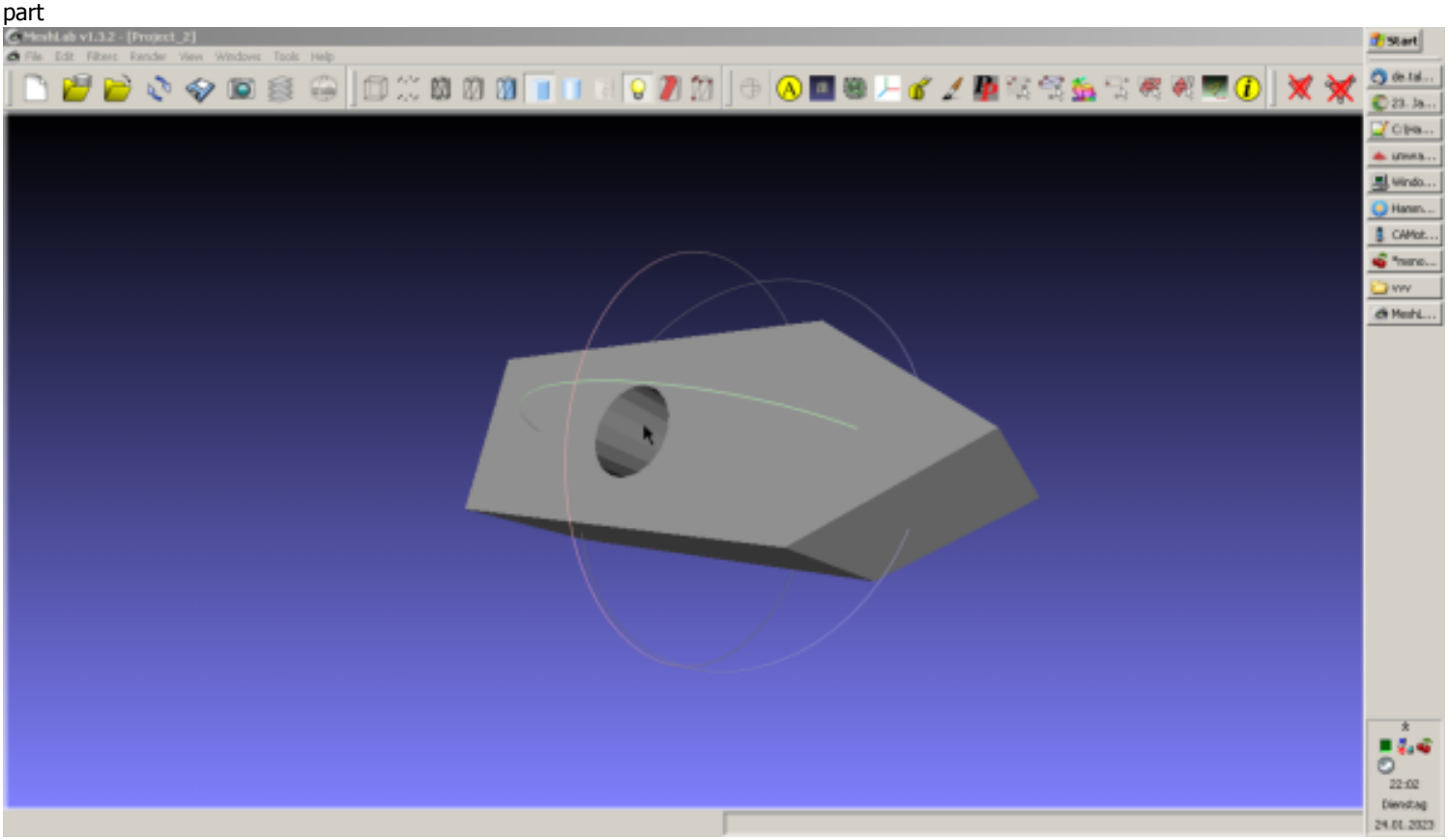
Due the leak of real 3D toolparths (six DOF) within GPLed Software the idea was born to unwrap a part with meshlab
https://www.meshlab.net/

Then create the prure “translational” (without rotations) toolpath with pycam.
https://pycam.sourceforge.net/

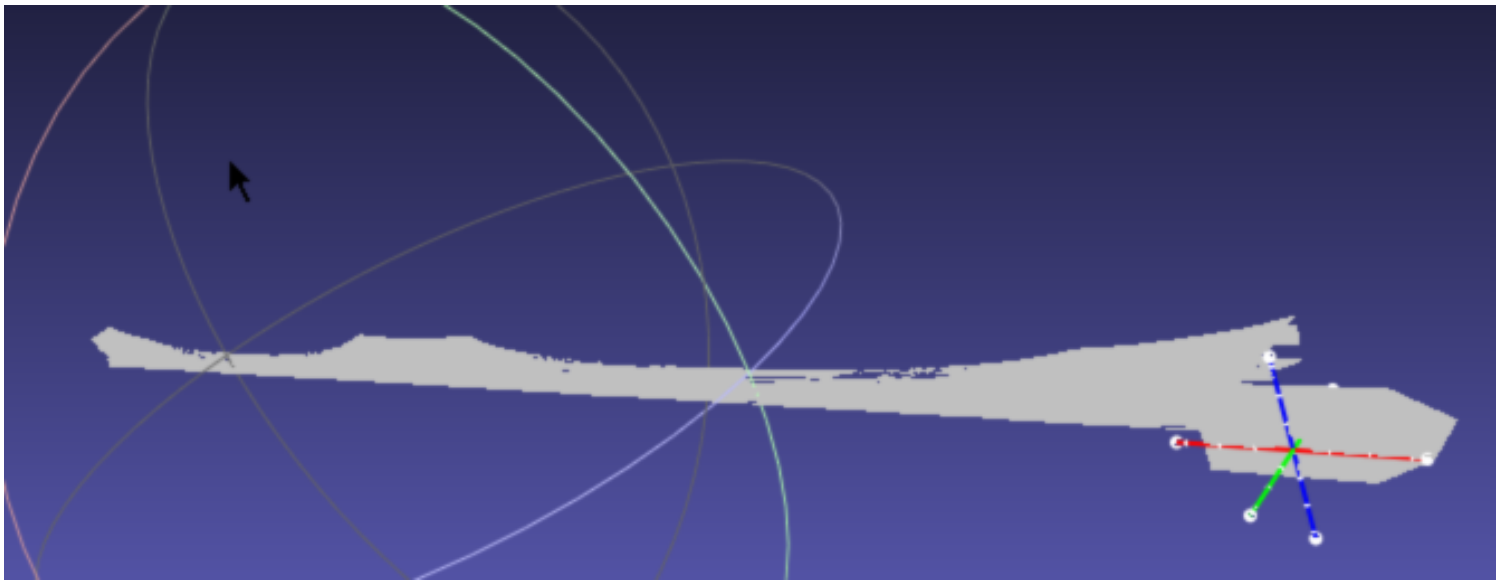
an after that, in a third step, wrap with Camotics/Toolpathlanguage:
https://camotics.org/
https://tplang.org/

Meshlab unwrapping (it does always around Y!)

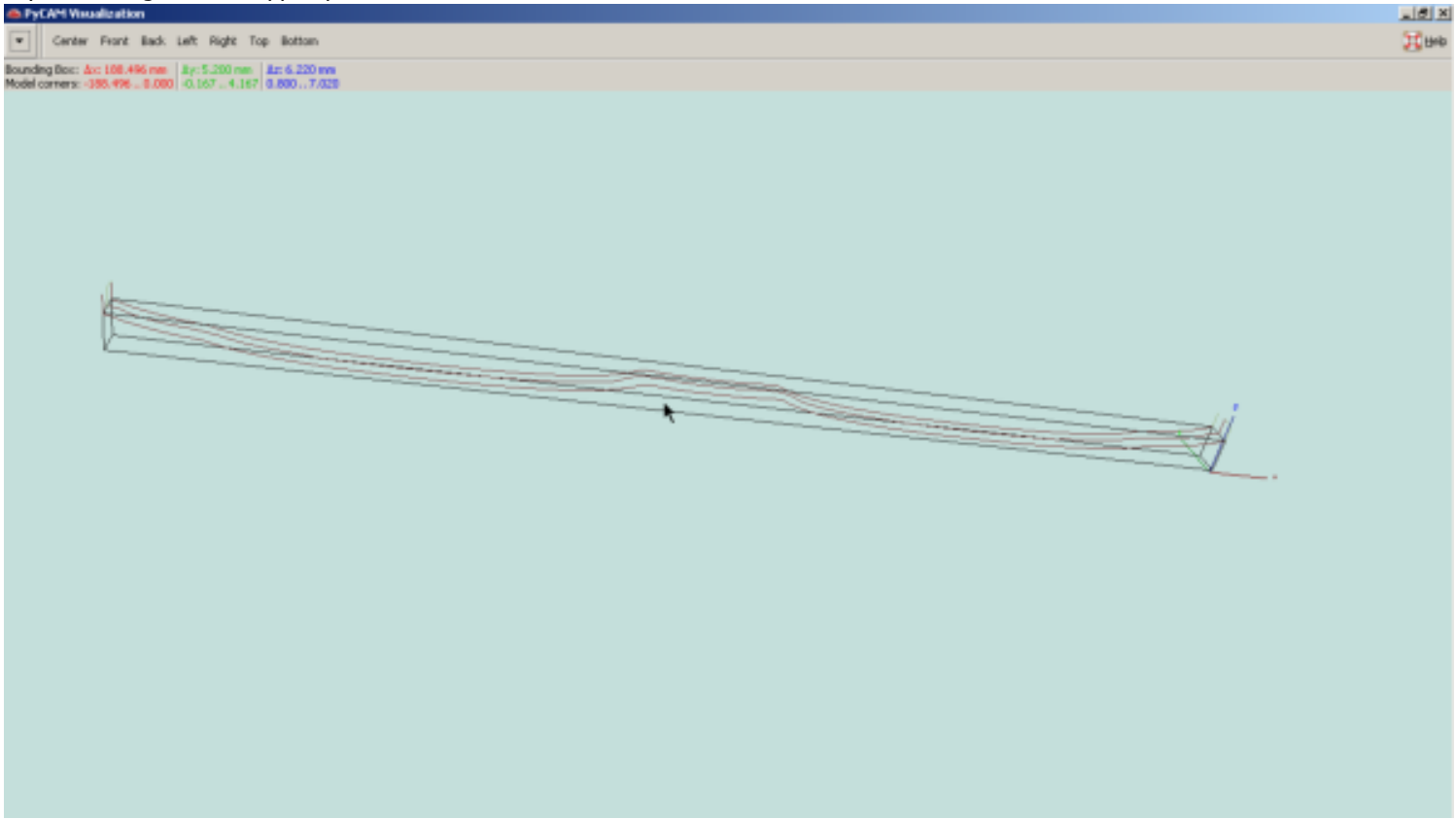
part



part and unwrap:



Pcycam milling the unwrapped part



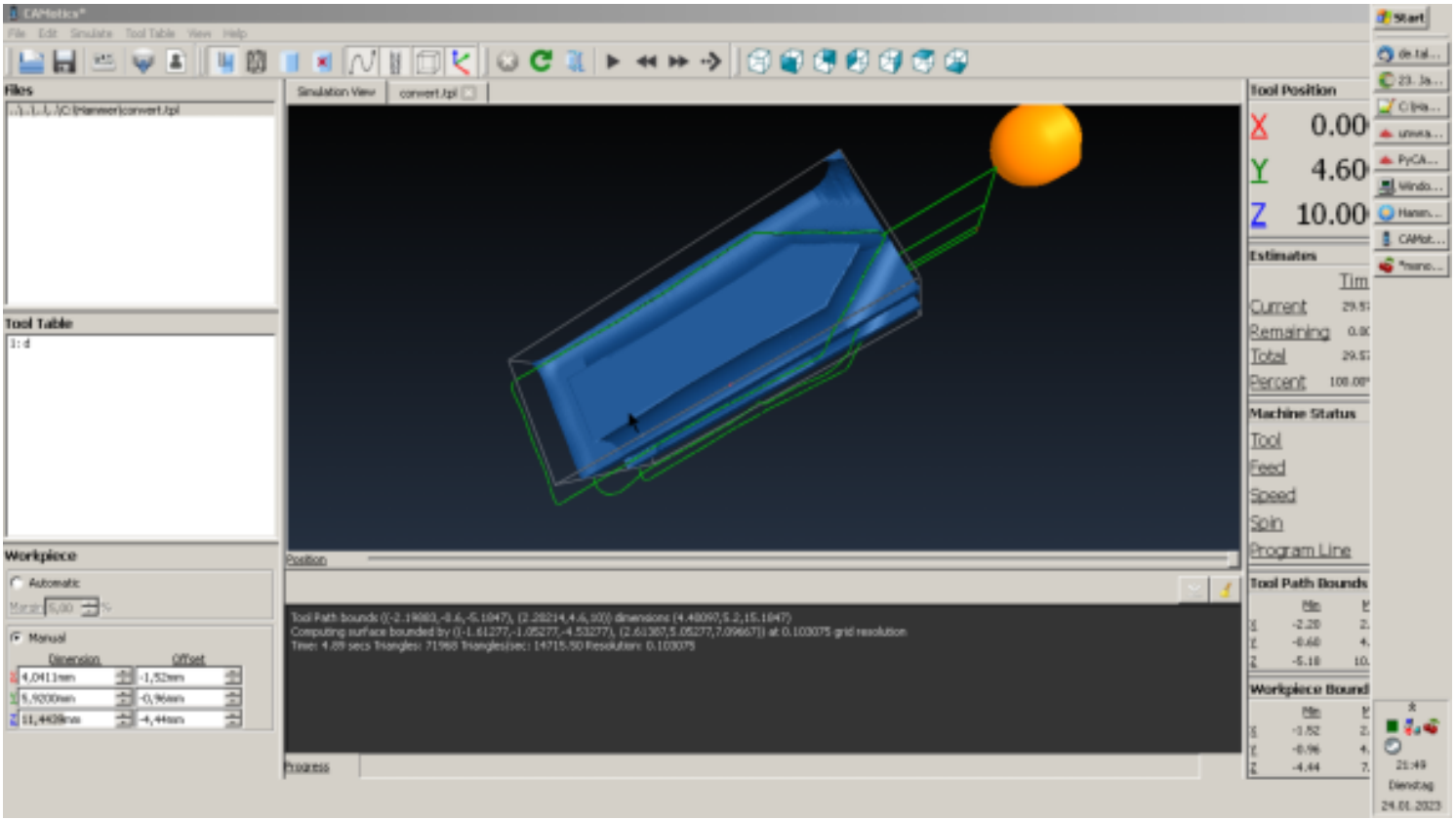
The Toolpath script:

```
units(METRIC);
size = 1;


//tool(1);
feed(3000);


// full is robocode, full position plus orientation at tcp
// setting it to false means rotation about y axis at worpiece zero
var fulltransform=false;


//wrapping Diameter fom meshlab
var D=30;


//cut around any rotational ax given by AX,BY,AZ
```

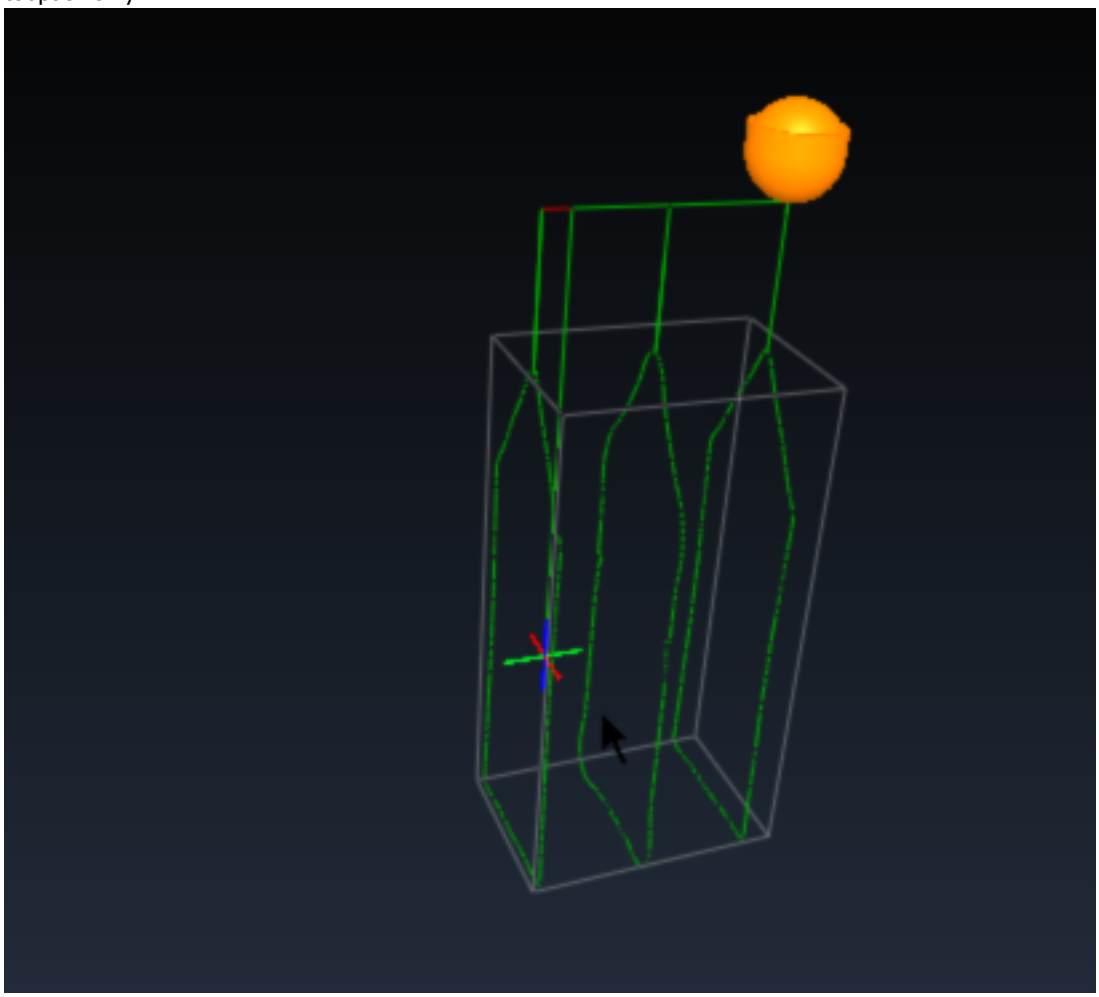
```
function rcut(o){
  if (isNaN(o.a)){cut(o);}else{
    loadIdentity();
    rotate(o.a/180*Math.PI,0,1,0);
    cut(o);
  }
}
//cut around any rotational ax given by AX,BY,AZ
function rrapid(o){
  if (isNaN(o.a)){cut(o);}else{
    loadIdentity();
    rotate(o.a/180*Math.PI,0,1,0);
    rapid(o);
  }
}


//cut wrapped..rolled (the inverse Meshlab) with Diameter D around Y
function rxcut(o){
  //D=30;
  if(fulltransform){
    if (isNaN(o.x)){rcut(o);}else{
      //loadIdentity();
      //translate(x=-o.x, y=0, z=0);
      A=(o.x/(D*Math.PI*2))*(360);
      o.x=0; //entweder hier nullen oder besser das translate
      o.a=A;
      //print(A+"\n");//Kontrollausgabe
      rcut(o);
    }
  }else{
    if (isNaN(o.x)){cut(o);}else{
      //loadIdentity();
      //translate(x=-o.x, y=0, z=0);
      A=(o.x/(D*Math.PI*2))*(360);
      o.x=0; //entweder hier nullen oder besser das translate
      o.a=A;
      //print(A+"\n");//Kontrollausgabe
      cut(o);
    }
  }
}
function rxrapid(o){
  //D=30;
  if(fulltransform){
    if (isNaN(o.x)){rrapid(o);}else{
      //loadIdentity();
      //translate(x=-o.x, y=0, z=0);
      A=(o.x/(D*Math.PI*2))*(360);
      o.x=0; //entweder hier nullen oder besser das translate
      o.a=A;
      //print(A+"\n");//Kontrollausgabe
      rrapid(o);
    }
  }else{
    if (isNaN(o.x)){rapid(o);}else{
      //loadIdentity();
      //translate(x=-o.x, y=0, z=0);
      A=(o.x/(D*Math.PI*2))*(360);
      o.x=0; //entweder hier nullen oder besser das translate
      o.a=A;
      //print(A+"\n");//Kontrollausgabe
      rapid(o);
    }
  }
}
```

display modes in Camotics:

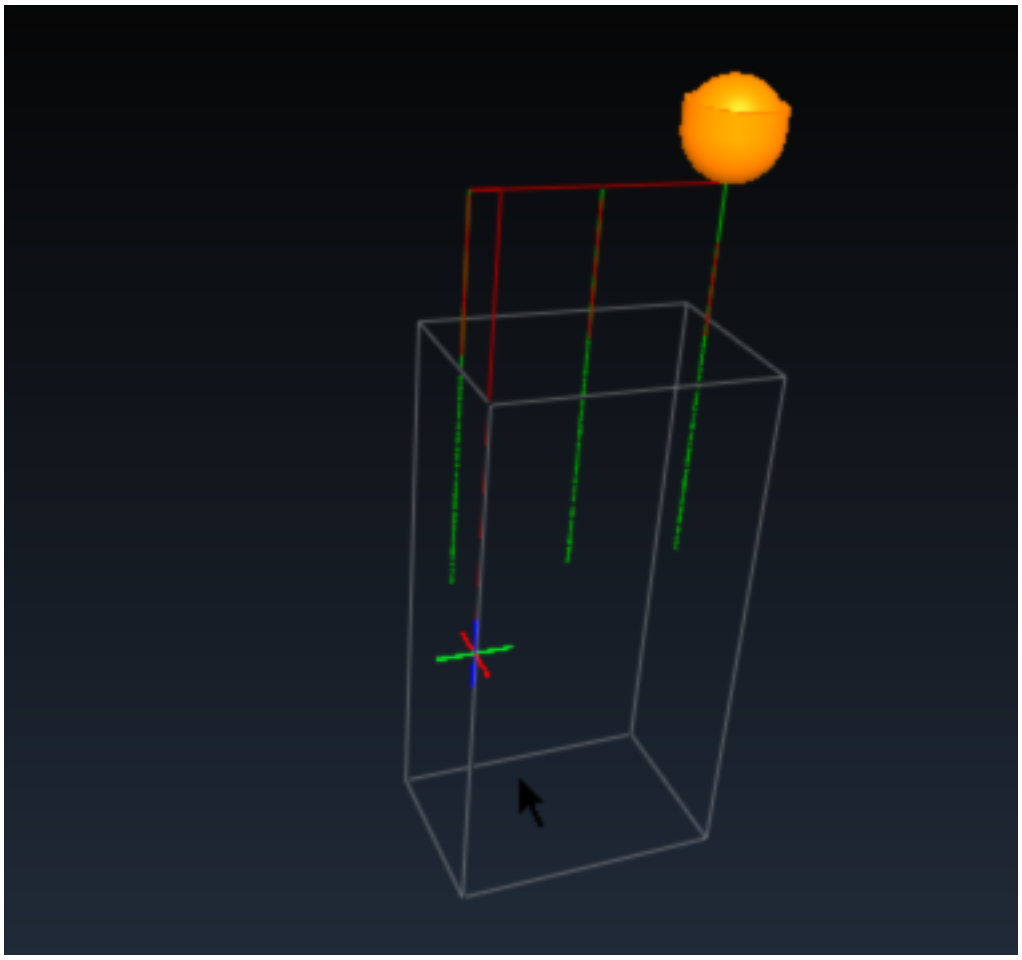
full Rotary with part:



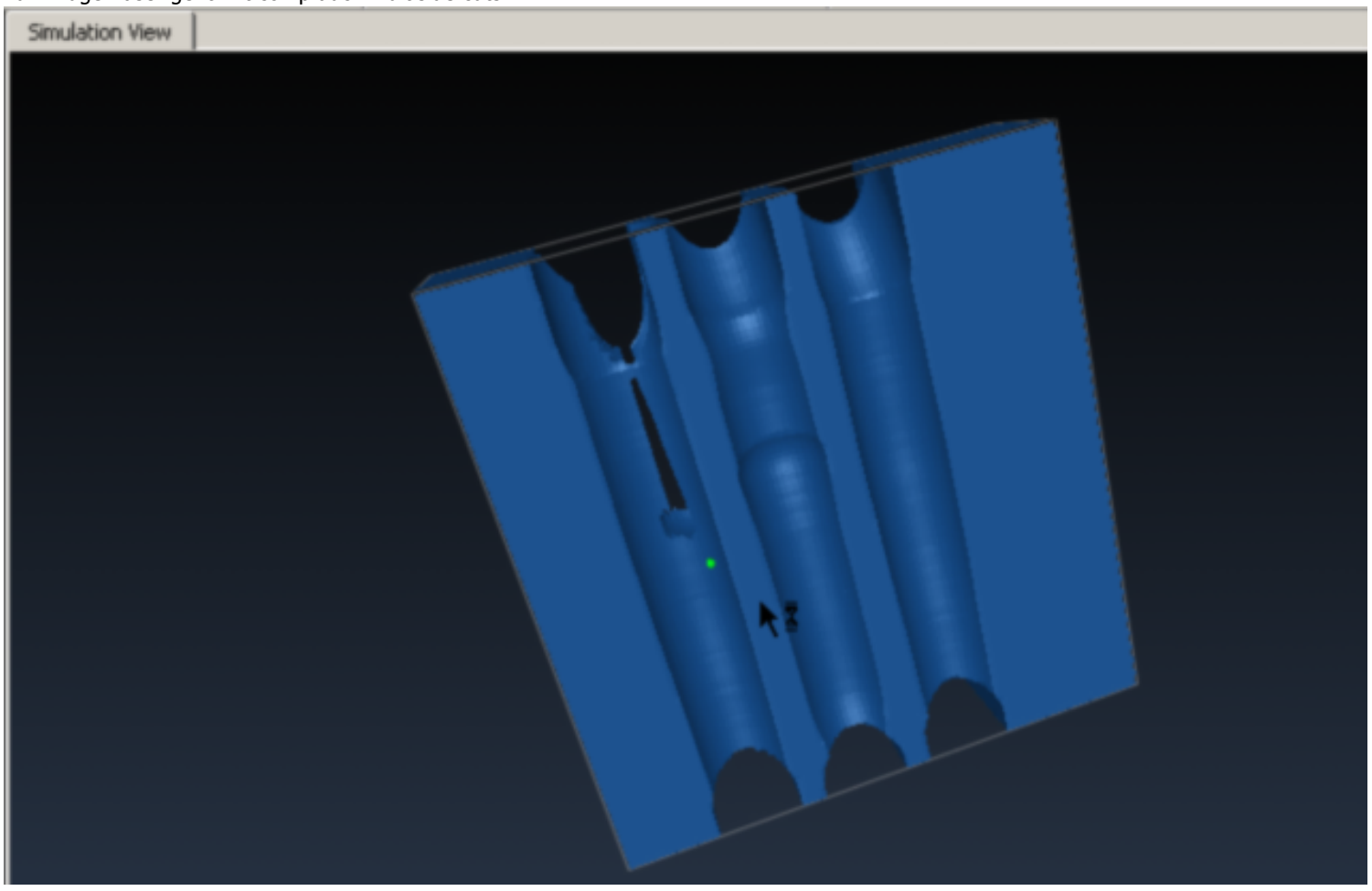
toopath only:



typical worpiecerotaion-view:

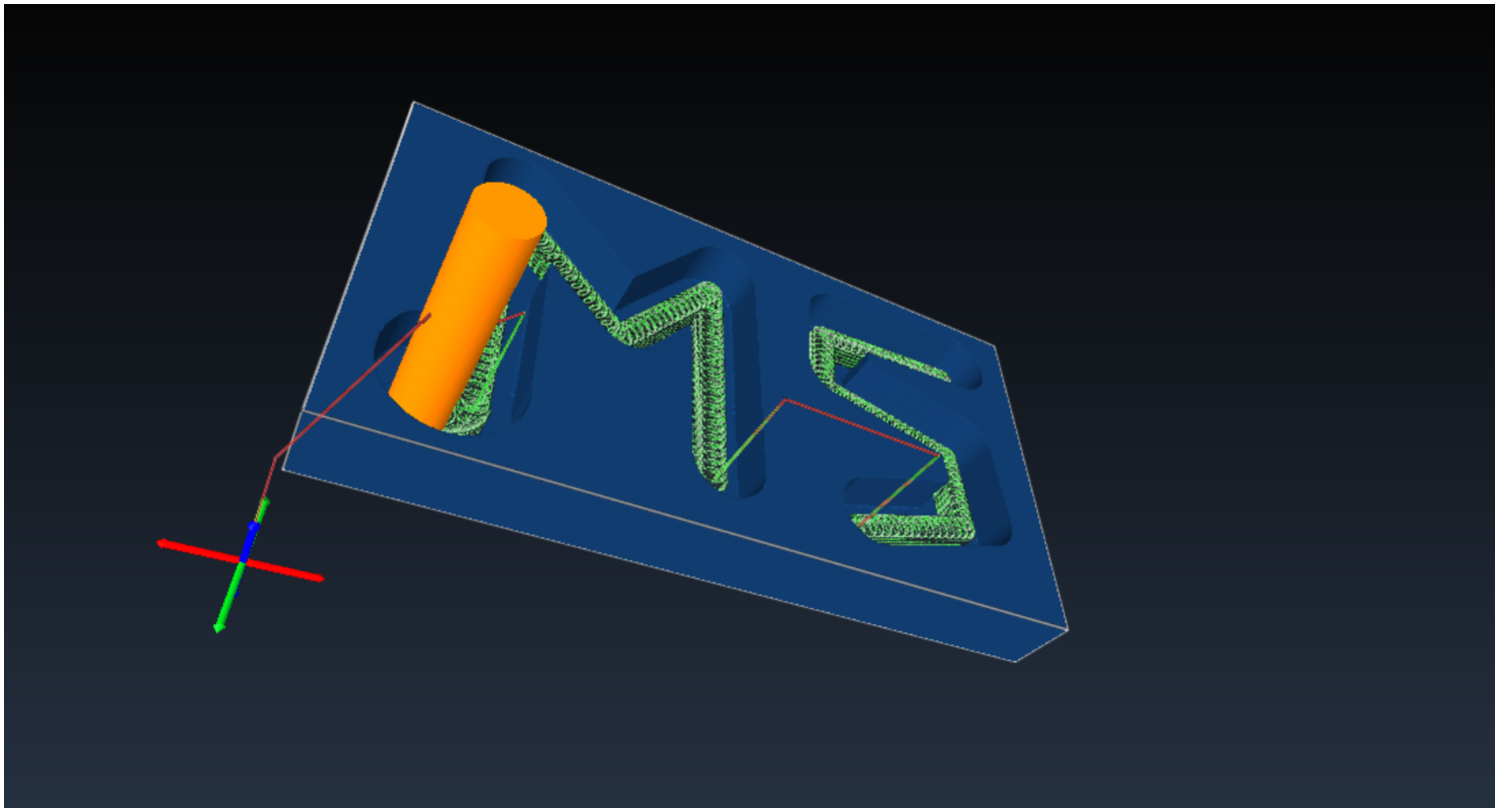


Fürn Kugelfräser geht mit comp auch Multiside cutsim!



Trpchoidal Fräsen

(eigene) Module im Camotics:



Full transforming rotary Camotics

Workflow stays as in “rotary camotics”, but

got sylvester.js as Matrix api integrated into camotics.

with this we can fully transform viewport from MachineZeropoint (no rotation) into WorkpieceZeropoint(Viewport rotatetes with workpiece) , swap axis a, wrap and rotate easily!!!

Codeparsing can completly be done here- or in an extra node (node.js)

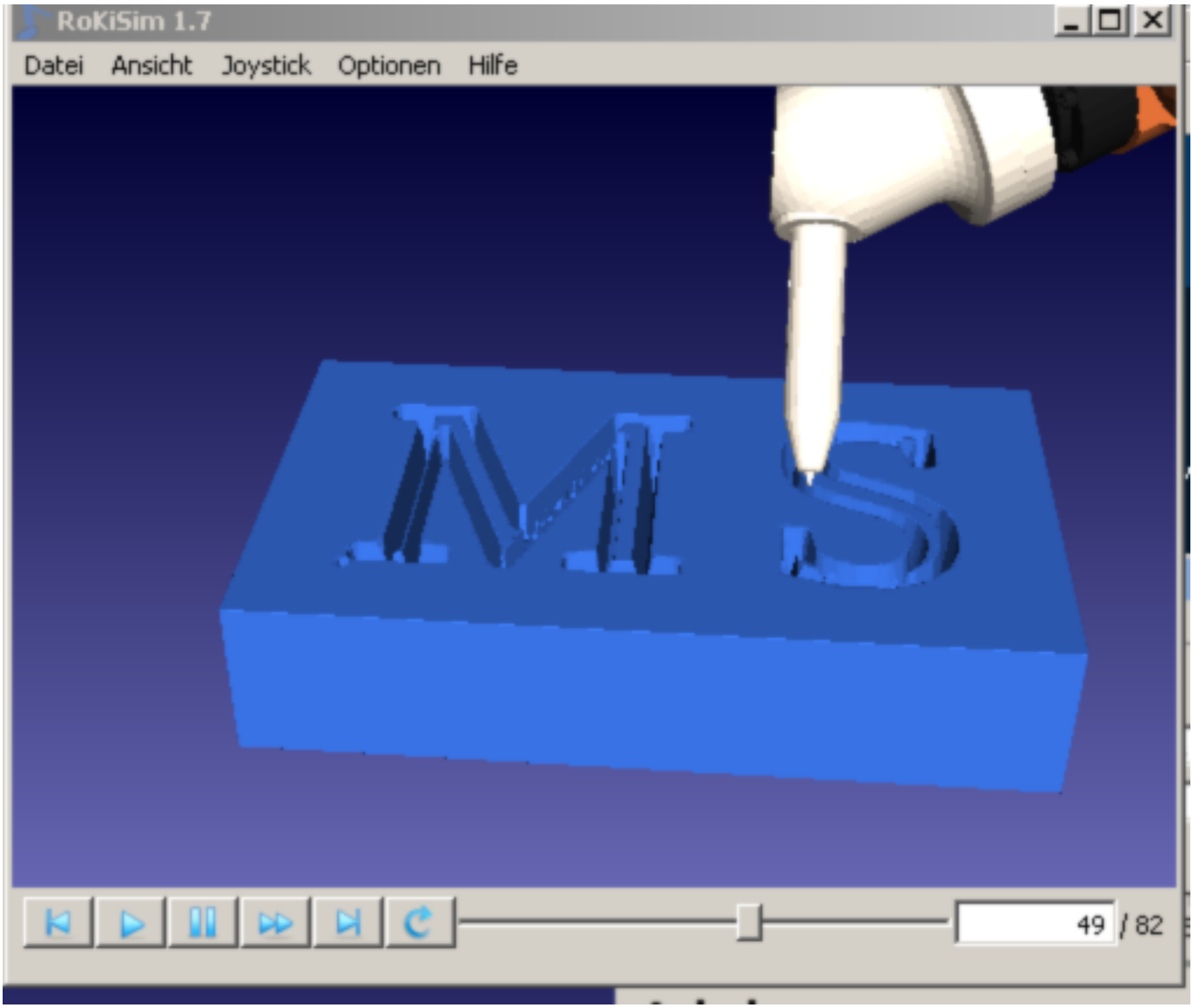
samples here:

Rokisim benutzt Achswerte zum Simulieren, der Parser hierfür hat die IK des Roboters zusätzlich zur matrix Api mit im Bauch!

Webb der Code passt(d.h. in Werkstückcoordinaten programmiert ist!") juckt das nicht--- hier wird das Ding zum Simulieren geignet!!! (Maschine egal, die wird nur zum parsen benutzt in Bsp einfach aus dem Viewport herausgeschoben!!)

Ob das Werkzeug von einem Roboter oder einer NC geführt wird, spielt hier keine Rolle!

Einfach TCP im WerkstückkoordinatenSystem programmieren!!!!



Für MontageAnlagen: Einfach TCP im MaschinenKoordinatenSystem programmieren.

Für Conti: Das Maschinenkoordinatensystem MUSS festgelegt werden! Konstrukteure!!!!

Toleranzen können automatisch vermessen werden: messpunkte (Winkel) an verschiebbare Komponenten schrauben und Taster an den Roboter!! (QR-Codes Messkamera?)

Beim einsetzen von Bauteilen: Werkstücknullpunkte BEIDE!!!! (für conti: Ventil und Block!!!!)

Matrix

TPLang Matrices

Basic Tool Path Generation

The basic functions of TPLang generate tool paths in the form of GCode. In particular the functions `rapid()` and `cut()` emit rapid and cutting moves respectively. For example, the following program generates a movement about a 1x1 square:

```
rapid(0, 0);
rapid(1, 0);
rapid(1, 1);
rapid(0, 1);
rapid(0, 0);
```

This will produce the following GCode:

```
G21     (mm mode)
G0 X1
G0 Y1
G0 X0
G0 Y0
%       (end of program)
```

Note, that a starting position of (0, 0) is assumed and GCode only requires writing the axes which have changed.

Basic Matrix Operations

The most basic matrix functions are `translate()`, `rotate()` and `scale()`. These functions can be used to apply transformations to the tool paths output by functions like `rapid()` and `cut()`. For example, if we take our previous program but first make a call to `translate()` the output is effected as follows:

```
translate(1, 1);

rapid(0, 0);
rapid(1, 0);
rapid(1, 1);
rapid(0, 1);
rapid(0, 0);
```

```
G21
G0 X1 Y1
G0 X2
G0 Y2
G0 X1
G0 Y1
%
```

Combining Matrix Operations

Now our 1x1 square has been translated to (1, 1). We can also scale and rotate as follows:

```
scale(3, 3);
rotate(Math.PI / 4);
translate(1, 1);

rapid(0, 0);
rapid(1, 0);
rapid(1, 1);
rapid(0, 1);
rapid(0, 0);
```

This produces a 3x3 square rotated 45 degrees and translated to (1, 1).

```
G21
G0 X1 Y1
G0 X3.12132 Y3.12132
G0 X1 Y5.242641
G0 X-1.12132 Y3.12132
G0 X1 Y1
%
```

Matrix operations accumulate via matrix multiplication. The default matrix is the identity matrix which does not change the tool path output at all. To get back to the identity matrix we can call ```loadIdentity()```. With this ability we can create square function and apply different matrix transformations to it. Of course it's not very interesting to do just rapid moves so let's try some cutting moves connected by a rapid.

```
function square(depth, safe) {
```

```
  rapid({z: safe});
  rapid(0, 0);

  cut({z: depth});

  cut(1, 0);
  cut(1, 1);
  cut(0, 1);
  cut(0, 0);

  rapid({z: safe});
}

feed(400);

scale(3, 3);
rotate(Math.PI / 4);
translate(1, 1);
square(-1, 2);

loadIdentity();
scale(2, 2);
translate(0.5, 0.5);
square(-1, 2);
```

Order of Matrix Operations

Matrix operations are combined through matrix multiplication. TPLang takes care of the details but it is important to note that unlike the multiplicate of real numbers order matters when it comes to matrx multiplication. Scaling then translating is not the same as applying the same operations in the reverse order.

```
scale(2, 2);
translate(1, 1);
square(zcut, zsafe);

loadIdentity();
translate(1, 1);
scale(2, 2);
square(zcut, zsafe);
```

Note that for the second square the starting position of the square (1, 1) is scaled up to (2, 2) because the translation is applied before the scale operation. The first square begins at (1, 1) as expected.

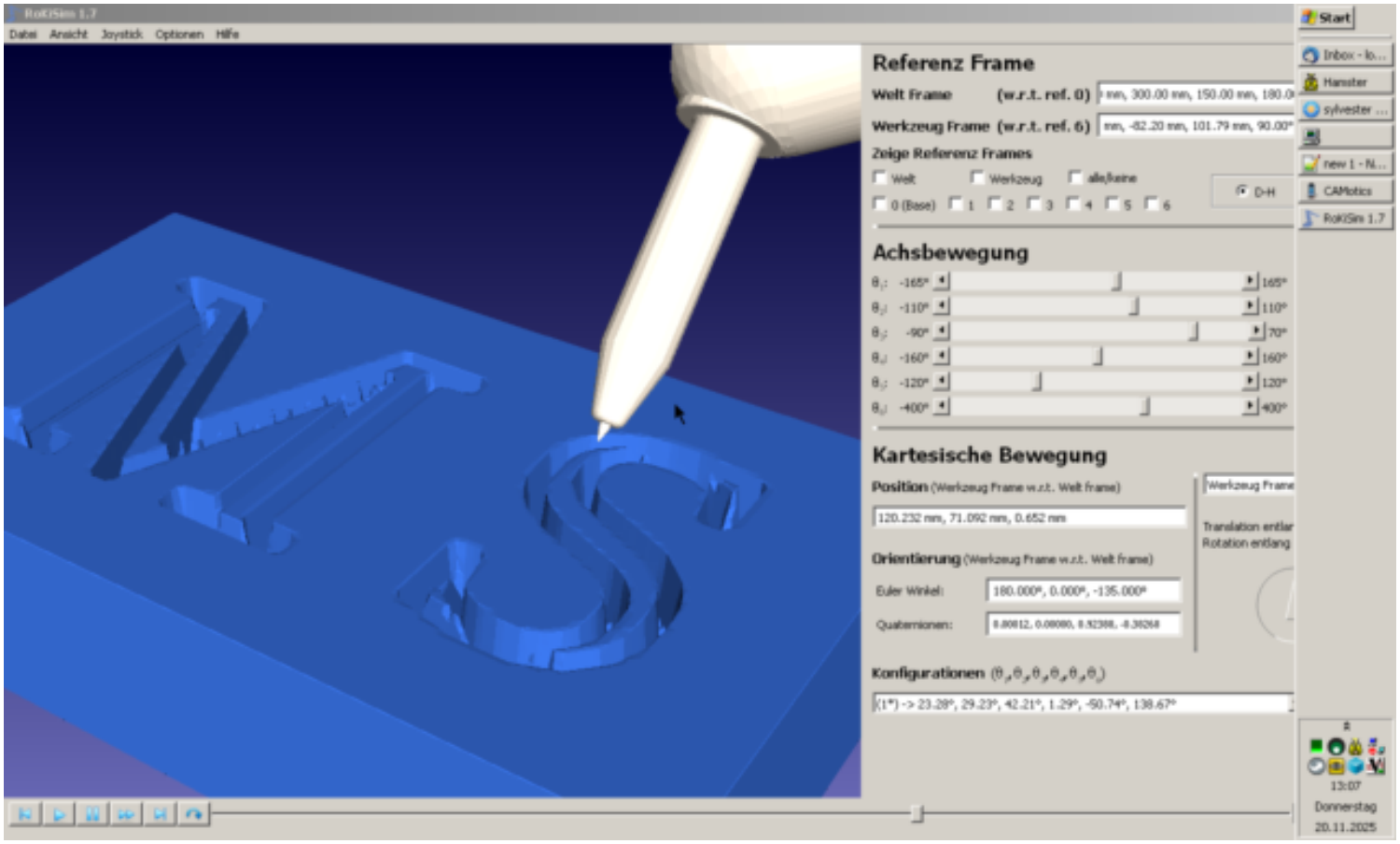
Matrix push and pop

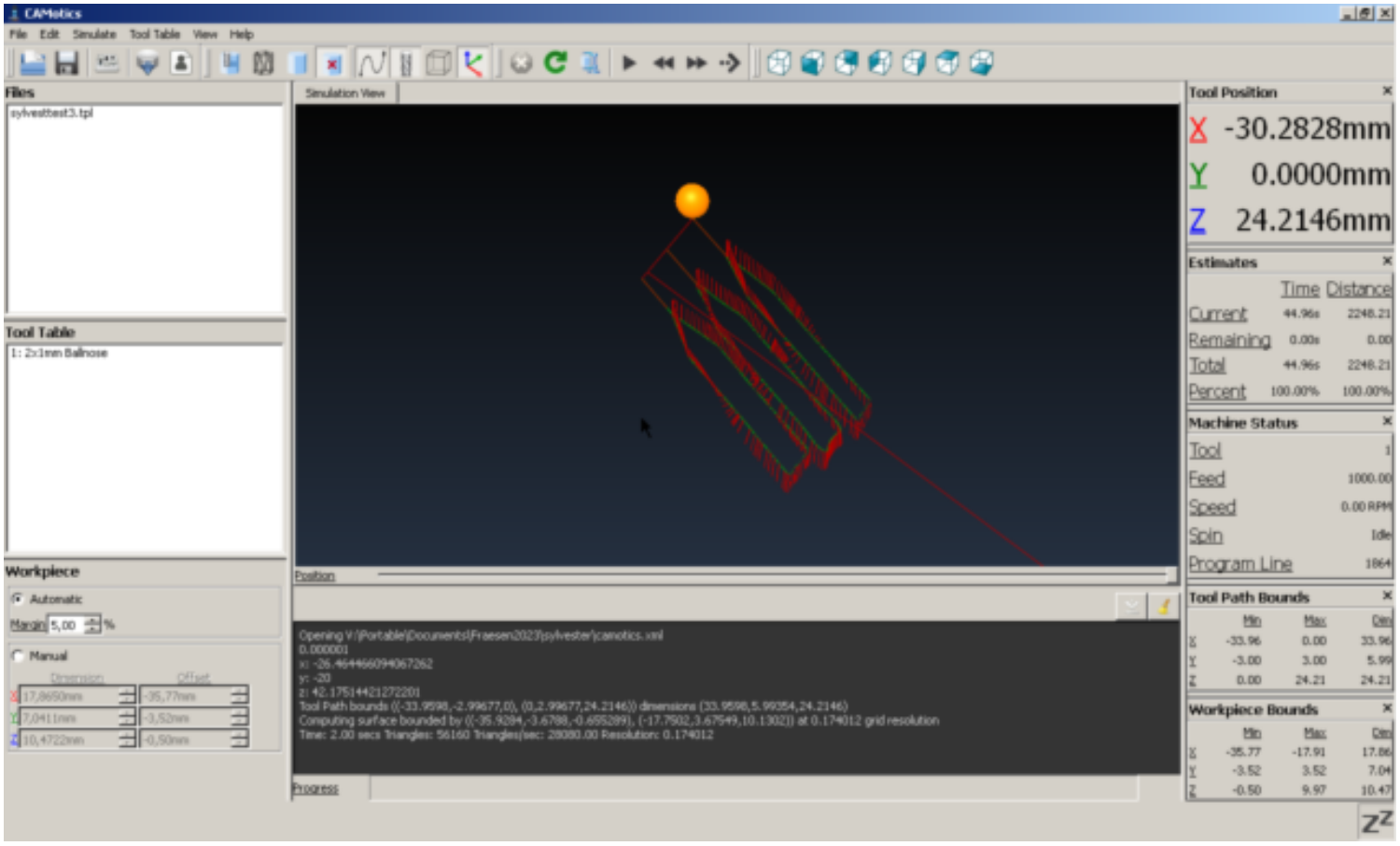
It is also possible to use `pushMatrix()` and `popMatrix()` to push and pop the current matrix to and from the matrix stack. This can be useful for applying temporary changes to the matrix.

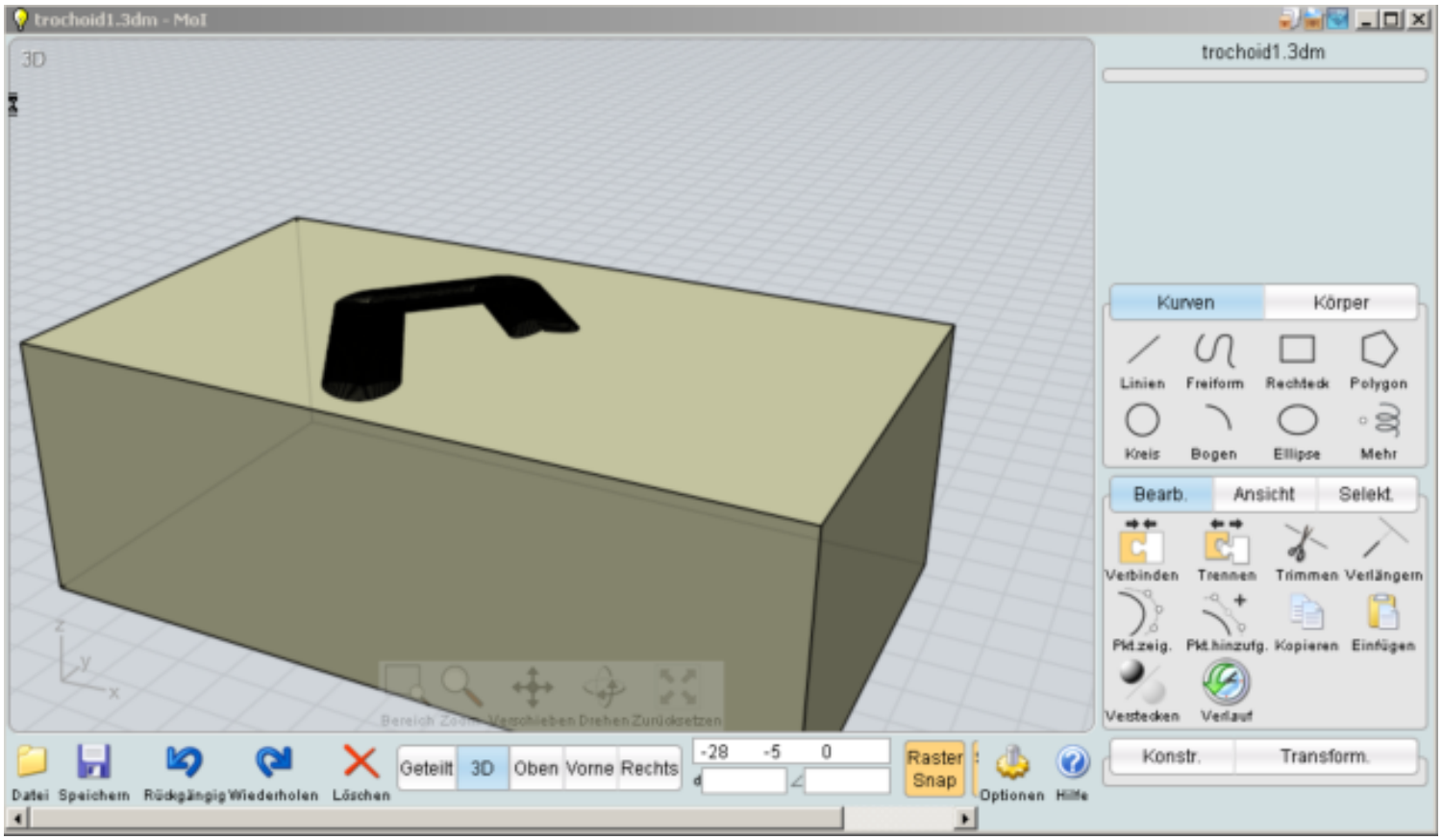
```
translate(1, 1);

pushMatrix();
scale(2, 2);
square(zcut, zsafe);
popMatrix();

square(zcut, zsafe);
```

Done- No more Diff between RobotnCNC

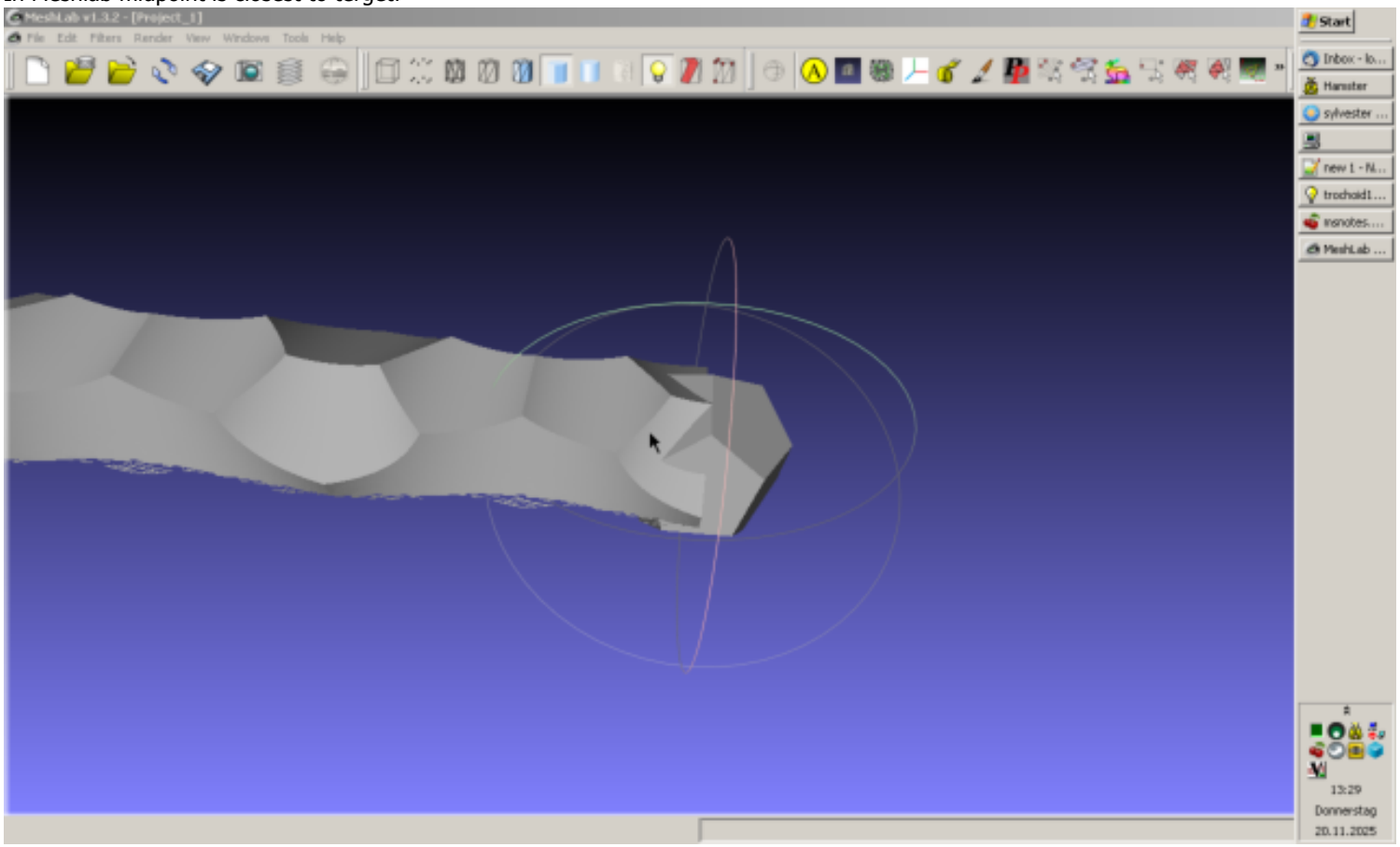
Even not in the screenS:



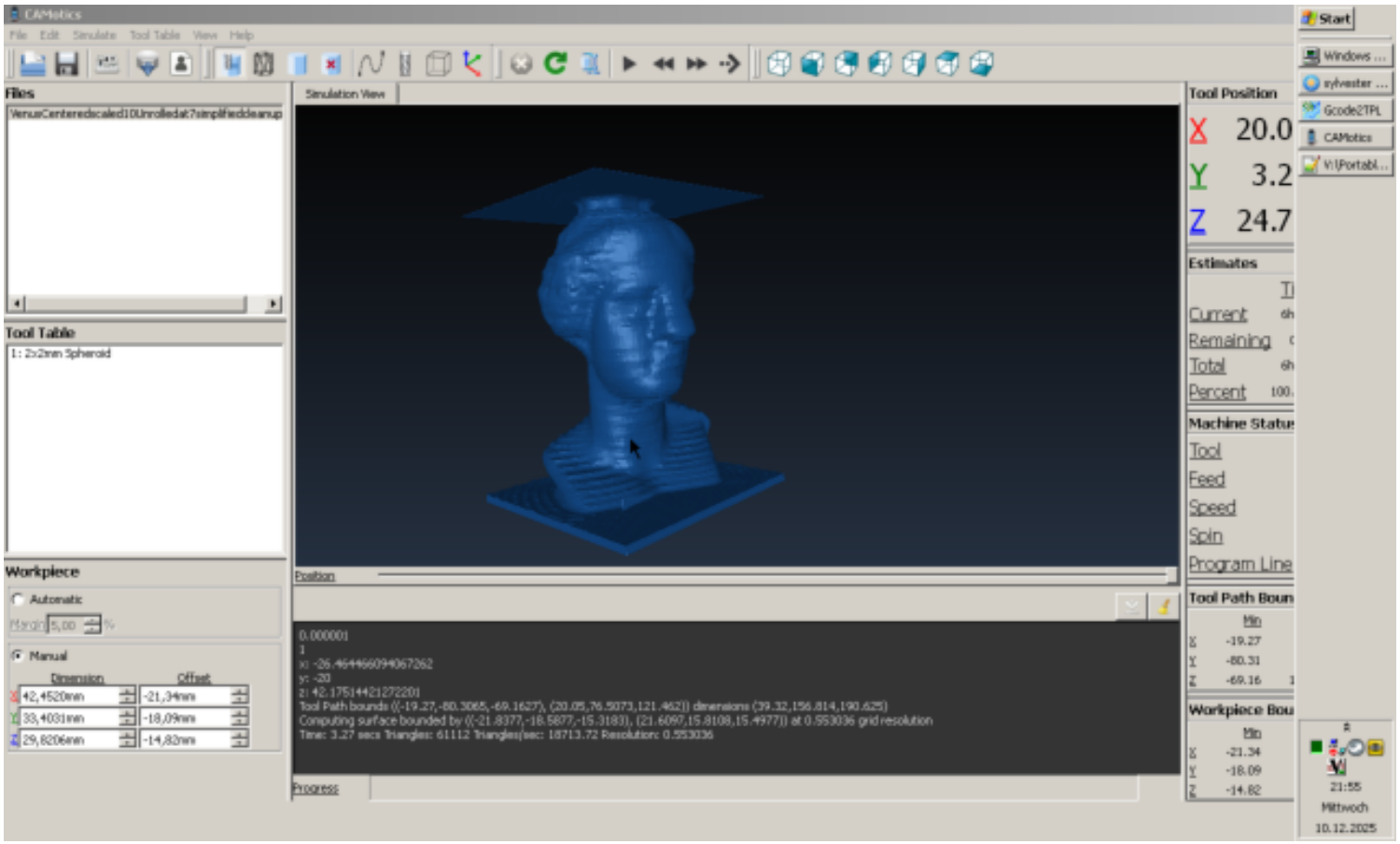




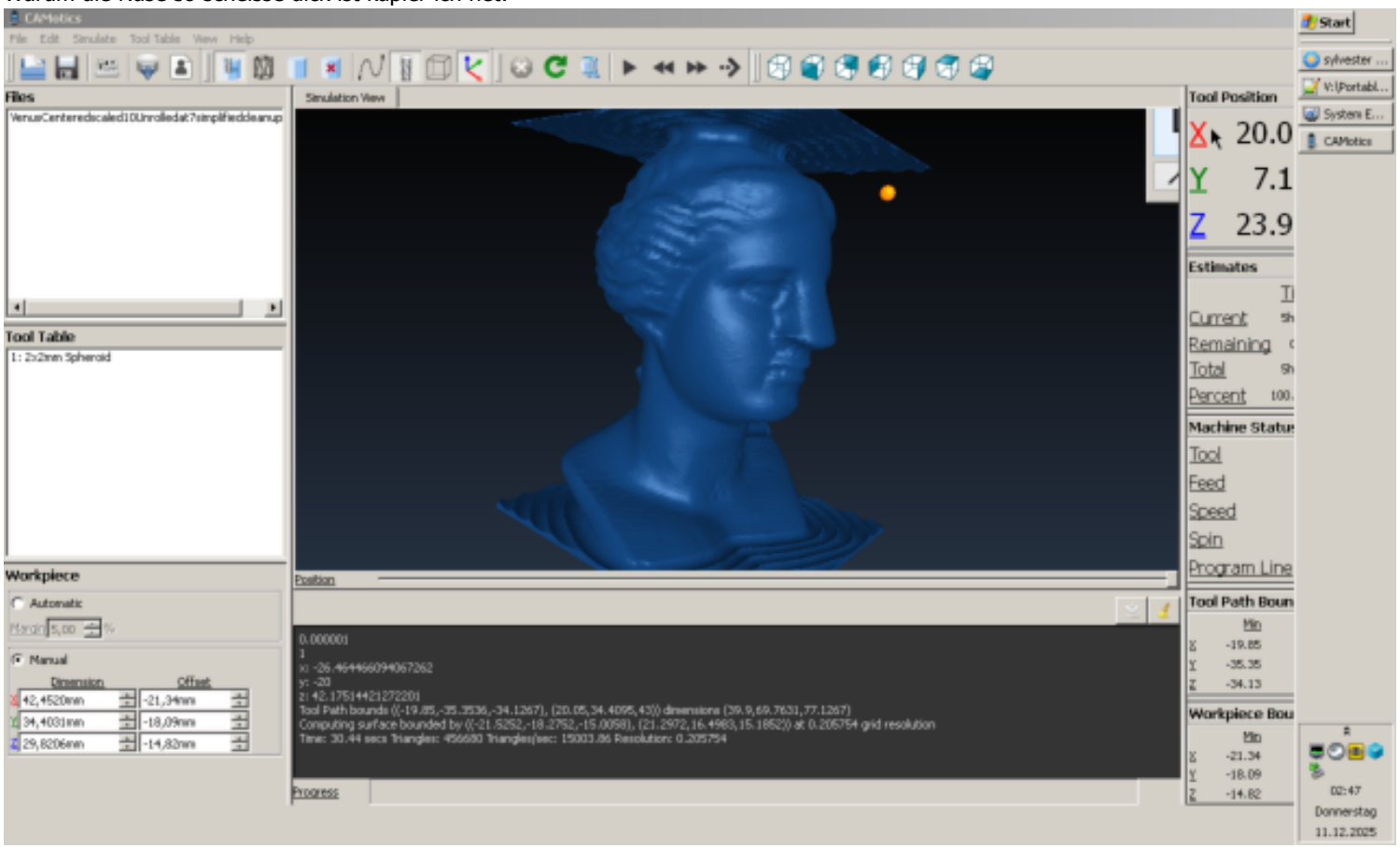
In Meshlab midpoint is closest to terget!



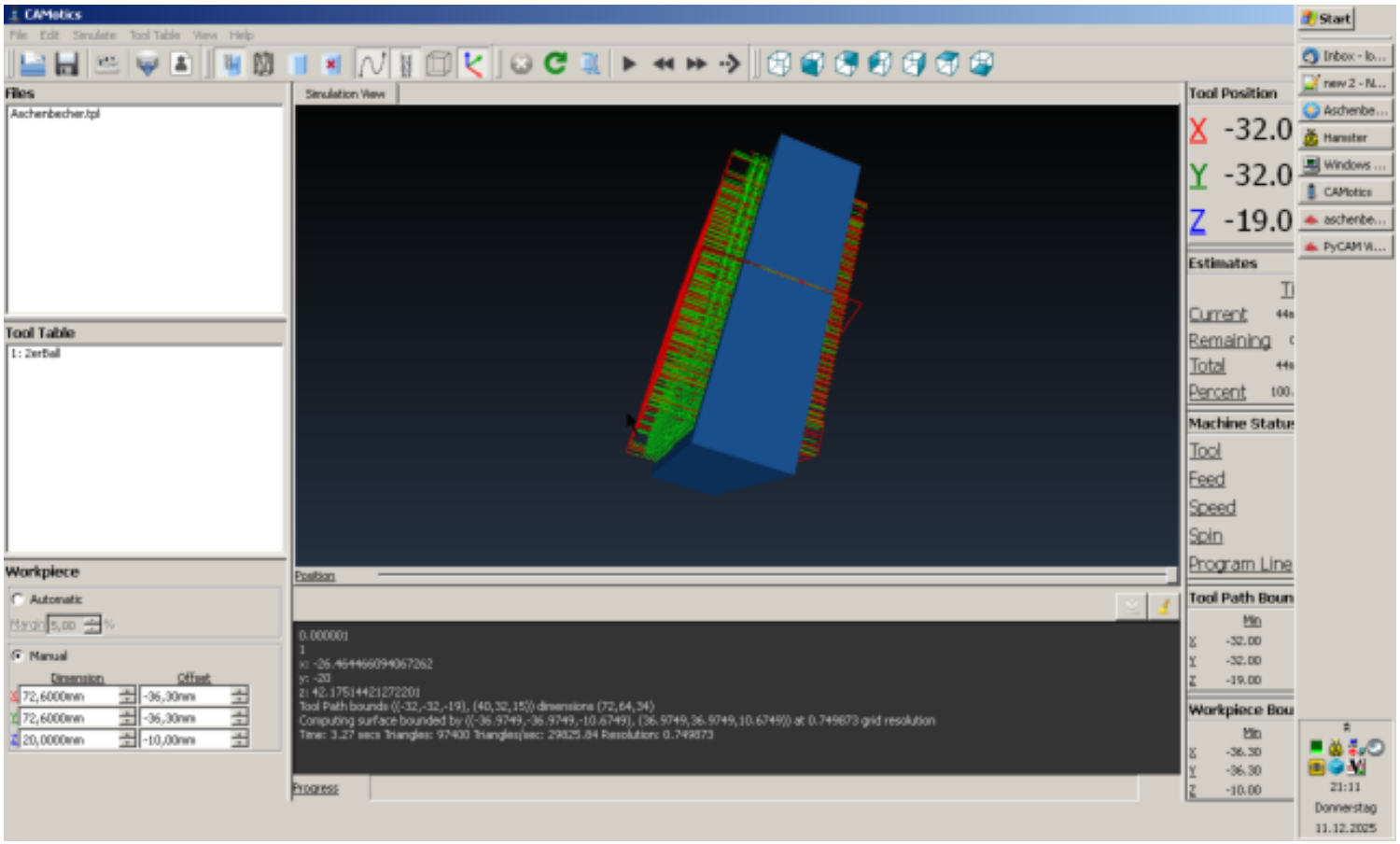
Venus Sample from Deskproto:

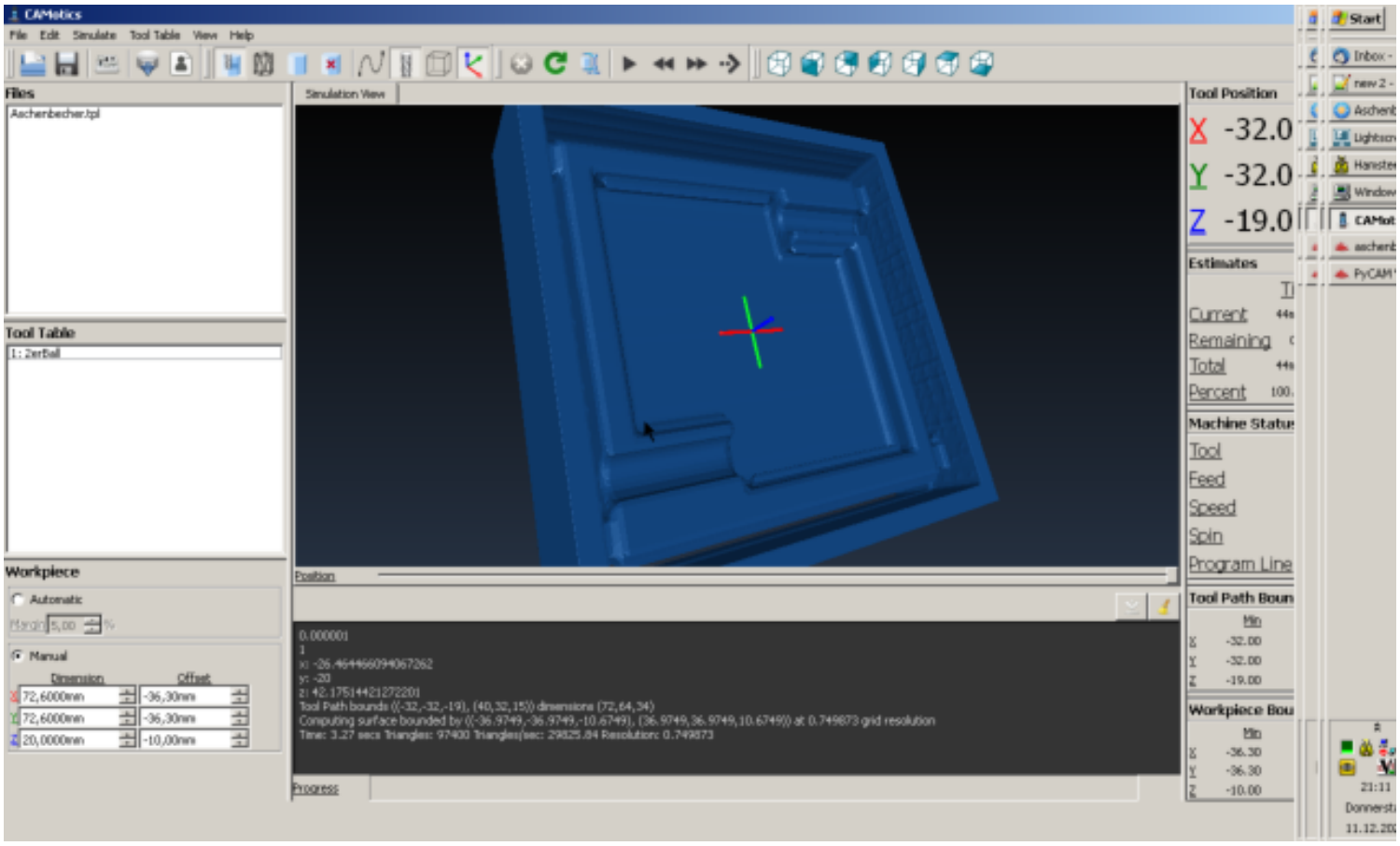


Warum die Nase so scheisse dick ist kapier ich net:



Das Klappt natürlich auch für Multiside! :


CAMotics
File Edit Simulate Tool Table View Help
Files
Aschenbecher.tpl
Tool Table
1: ZerBall
Workpiece
Automatic
Manual
Dimension
Offset
72,6000mm
-36,30mm
72,6000mm
-36,30mm
20,0000mm
-10,00mm
Simulation View
Position
Progress
Tool Position
X -32.0
Y -32.0
Z -19.0
Estimates
Current
Remaining
Total
Percent
Machine Statu
Tool
Feed
Speed
Spin
Program Line
Tool Path Boun
Min
Workpiece Bou
Start
21:11
Donnerstag
11.12.2025


CAMotics
File Edit Simulate Tool Table View Help
Files
Aschenbecher.tpl
Tool Table
1: ZerBall
Workpiece
Automatic
Manual
Dimension
Offset
72,6000mm
-36,30mm
72,6000mm
-36,30mm
20,0000mm
-10,00mm
Simulation View
Position
Progress
Tool Position
X -32.0
Y -32.0
Z -19.0
Estimates
Current
Remaining
Total
Percent
Machine Statu
Tool
Feed
Speed
Spin
Program Line
Tool Path Boun
Min
Workpiece Bou
Start