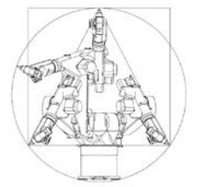
Introduction to **GRASSHOPPER** using KUKA|prc

parametric robot control for grasshopper

Association for
Robots in Architecture



Johannes Braumann & Sigrid Brell-Cokcan | Association for Robots in Architecture: 04/2012

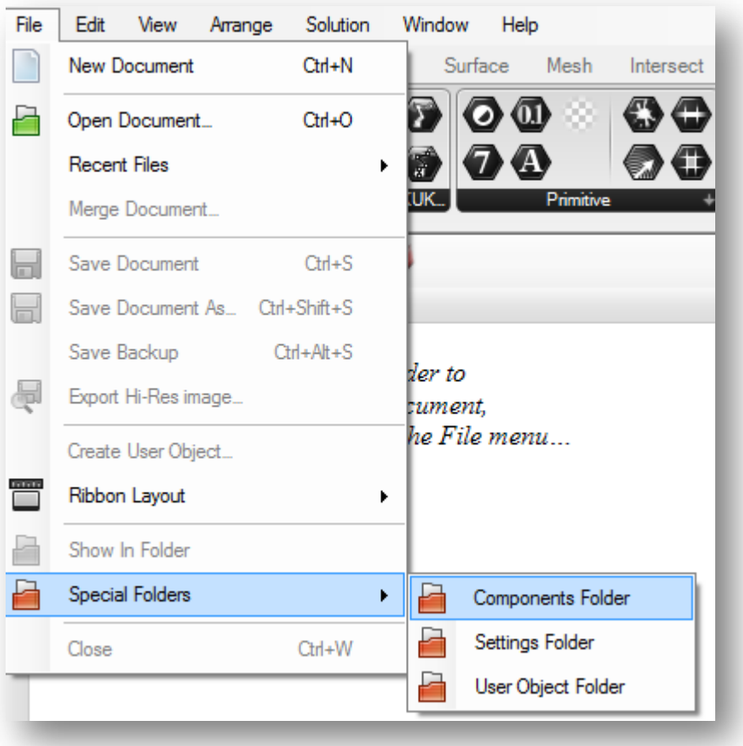
01 | Installing KUKA|prc

| Download KUKA|prc from www.robotsinarchitecture.org/kuka-prc

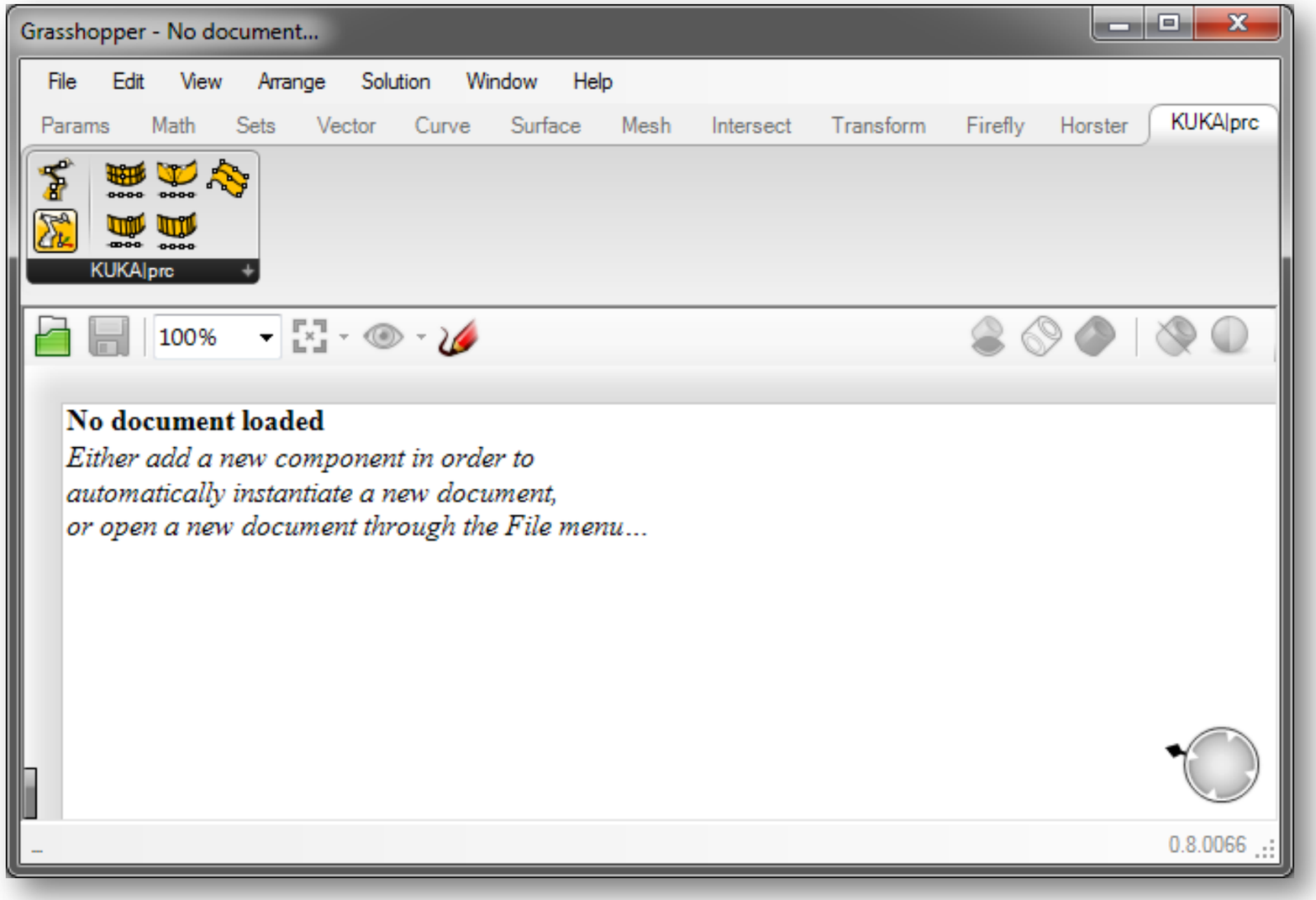
| Launch Grasshopper by typing “Grasshopper” in the Rhino command prompt

| In Grasshopper, click on File/Special Folder/Components Folder

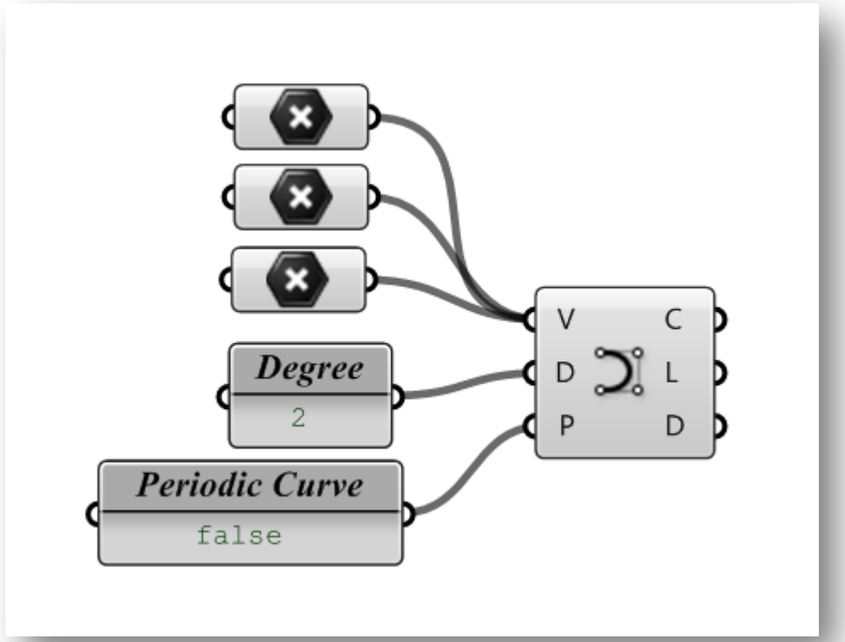
| Copy the contents of the downloaded file into this folder. Restart Rhino

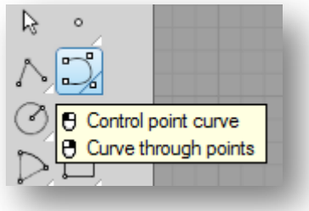


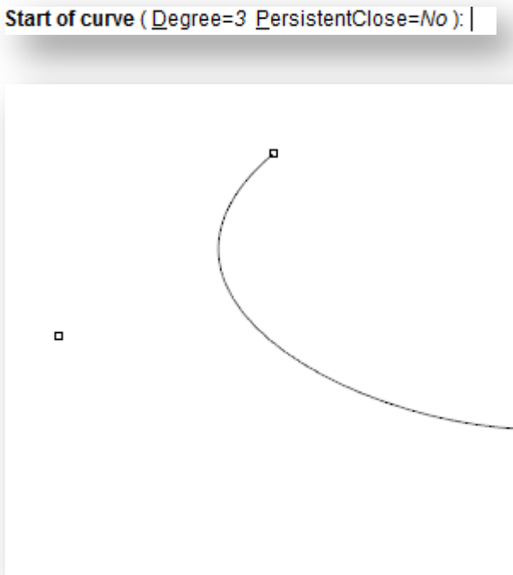
| The KUKA|prc tab should now show up in the Grasshopper menu.
It should contain the 7 icons as below.



| Grasshopper is a *PLUGIN* for Rhinoceros. It uses similar operations/parameters and shares the viewport with Rhino. GH-curve definition (left), Rhino curve creation (right).



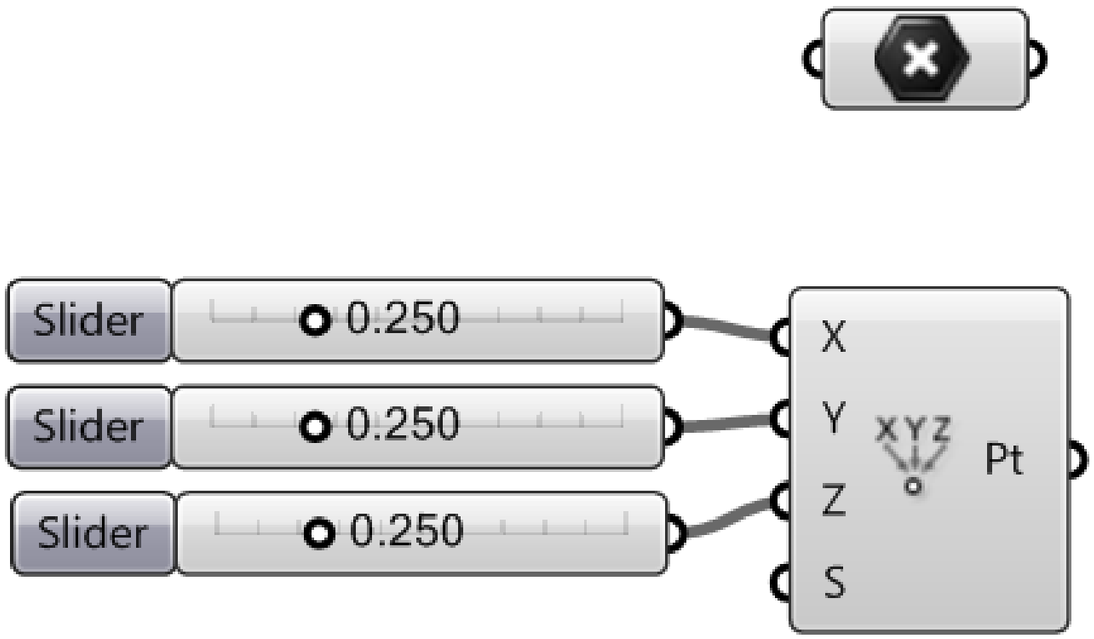




| Concept of Grasshopper is that you inform components, and pass information by linking components with each other.

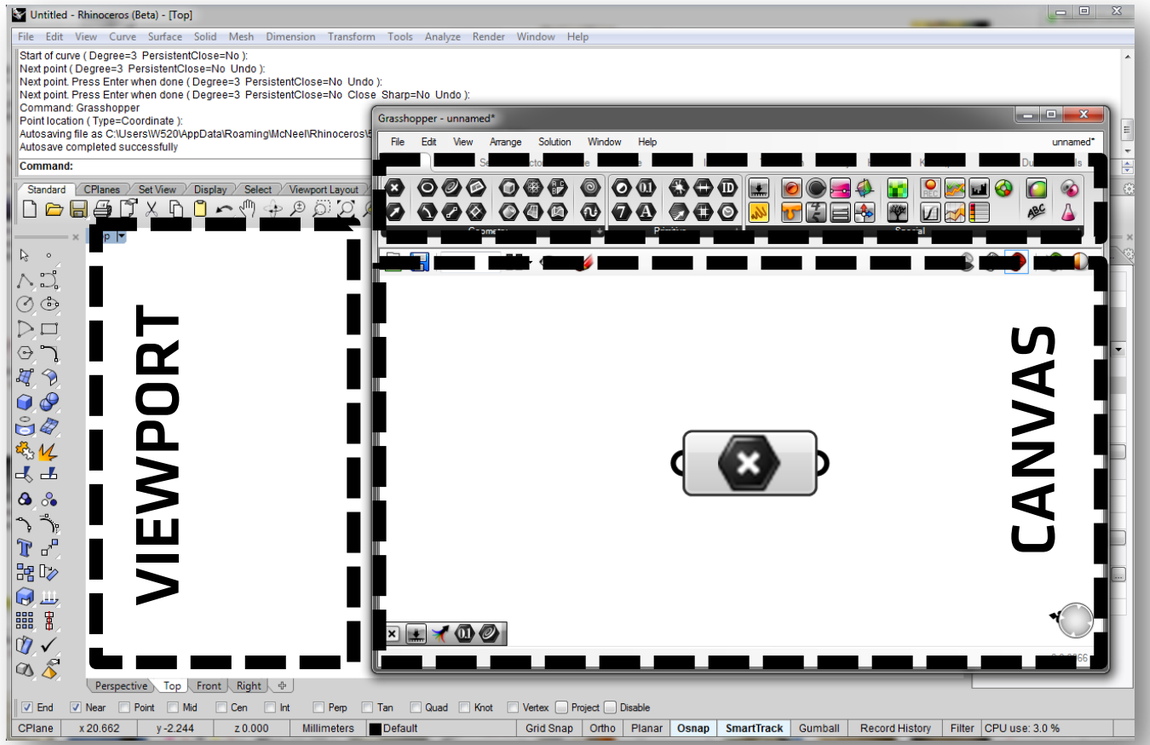
| Information can either come from inside GH, or by linking data directly from Rhino.

| We'll now start by creating a simple point, first using data from Rhino, and then from Grasshopper.



| Start by launching Grasshopper.

| We will refer to the Grasshopper icon groups as *tabs*. They get placed in the Grasshopper *canvas*. Geometry will show up in the Rhinoceros *viewport*

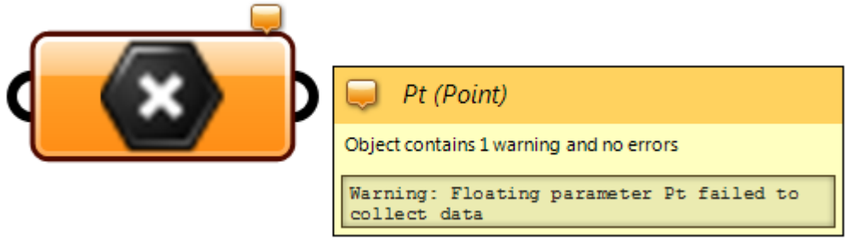


| In Grasshopper, open the Params tab and then click on the “Point” icon in the upper left corner. Drag it onto the Grasshopper canvas.

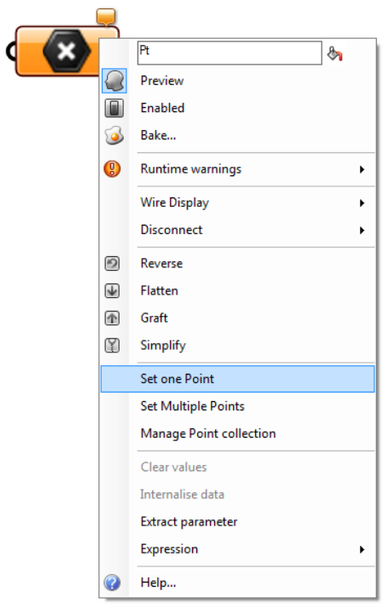
| The icon will show up in yellow. The following color-codings exist:

Grey – the component is working as intended
Yellow - the component is empty or that there are minor problems
Green – the component is selected
Red – there are serious problems with the component
Dark Grey – the component is hidden
Pale Grey – the component is disabled

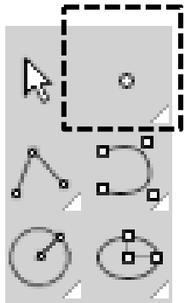
| When you put the cursor over the balloon of colored components, GH will list the problem. Our point says:



| The problem is that the component is empty. To fill it with data from Rhinoceros, right-click it and choose “Set One Point”

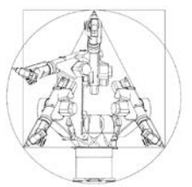


| You can now place a point in the Rhino viewport. It is also possible to reference existing geometry. To do so, switch to Rhino and use the Point command to place a few points.



| Back in Grasshopper, right-click again on the Point component and choose again “Set one Point”. However, in Rhino go to the command bar and click on <Coordinate>. This will show the following choice. Click on Point.

Grasshopper Point type <Coordinate> (Coordinate Point Curve):

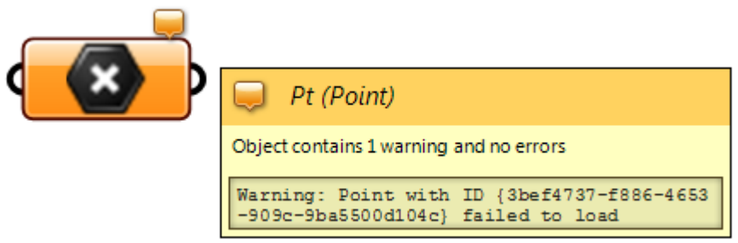


| The point is now referenced in Rhino and a red cross ✖ will be overlaid over the regular Rhino-point object.

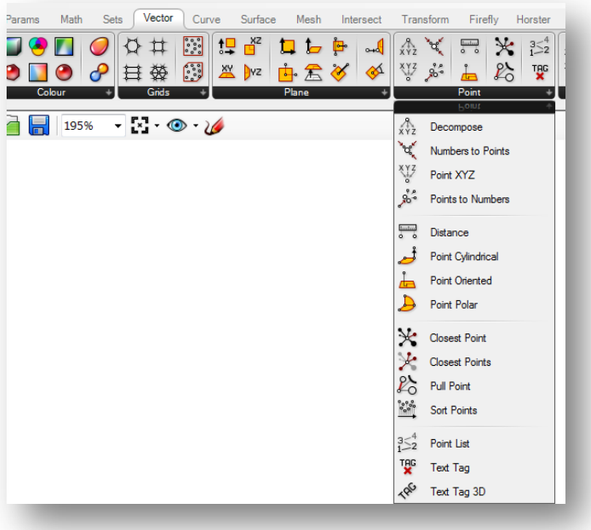
| When a Grasshopper component is selected, both the component and the contained geometry will turn green:  ✖

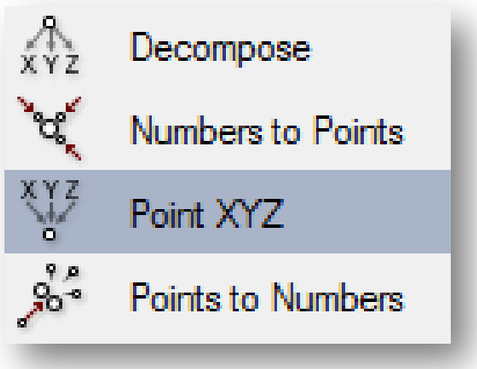
| Whenever you move a referenced object, the linked Grasshopper object will update as well. Move a point!

| If you delete an object that is referenced in Grasshopper, the GH component will turn yellow to show the error:



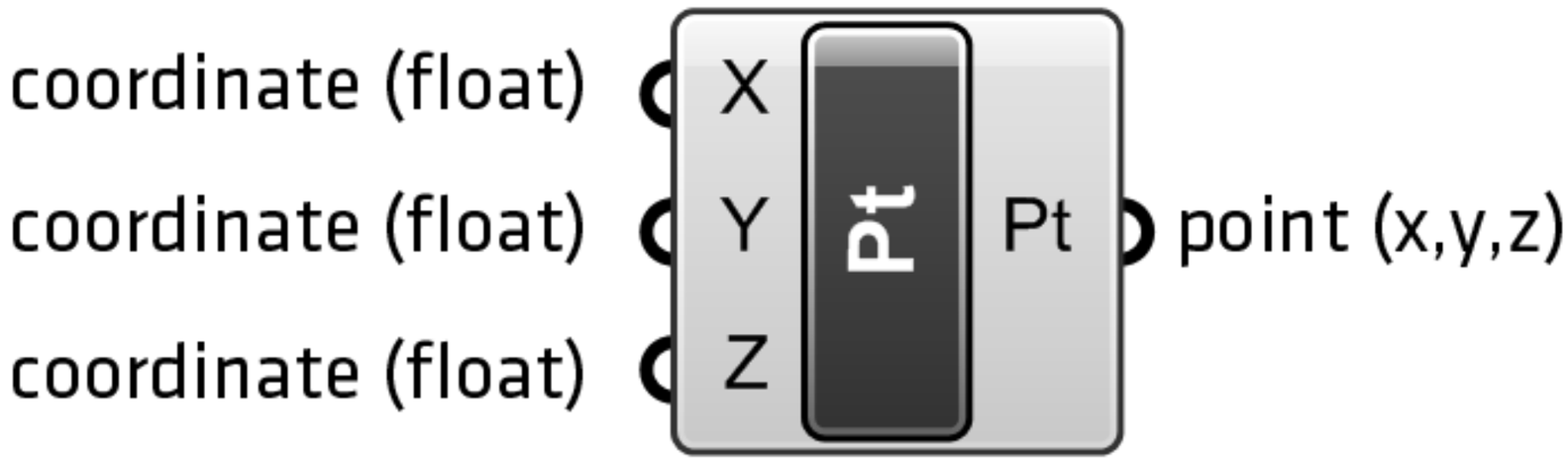
| We will now create a point inside Grasshopper. In GH, go to the Vector tab and then to the Point subsection. Click the small arrow on the lower right so that all components are shown.



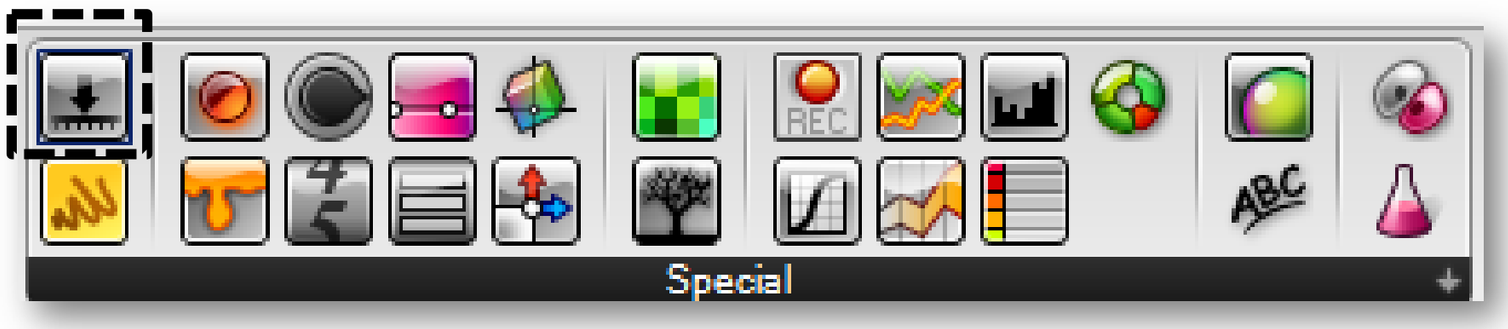


| Choose “Point XYZ” and drag the component onto the canvas.

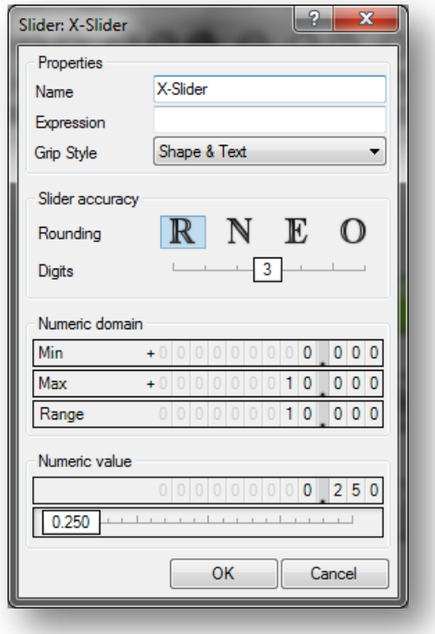
| Take a look at the component and mouse over the inputs. You will see that the Point XYZ component requires three numbers (XYZ) and outputs a point.



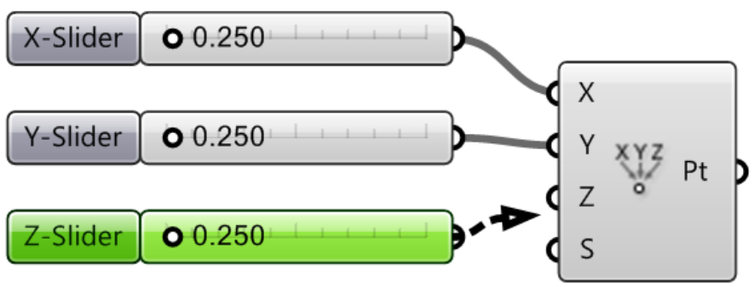
| Let's create some slider to create the XYZ values. You can find the Number Slider component in the Params tab, in the upper left corner of the Special subsection. Drag three slider onto the canvas.



| You can set the properties of the slider by double-clicking its name. Call the first slider “X-Slider” and set its range to 0.0-10.0. Repeat it for the remaining to sliders, Y-Slider and Z-Slider.



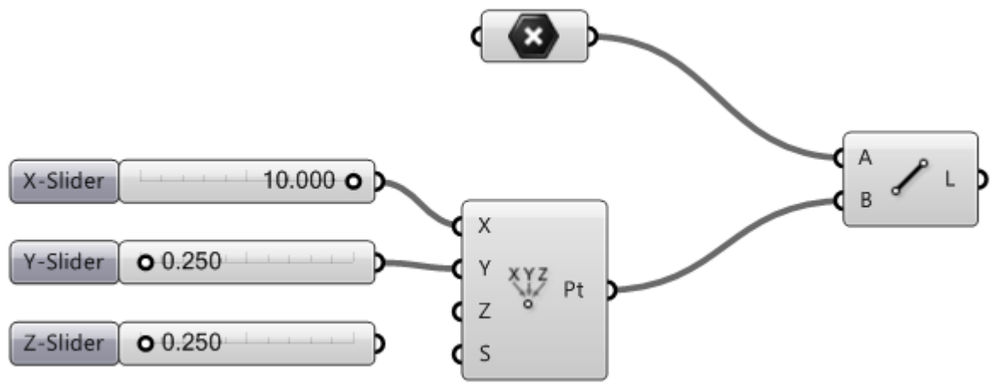
| Now connect the slider output of the slider into the corresponding input of the Point component. Do this for all sliders.

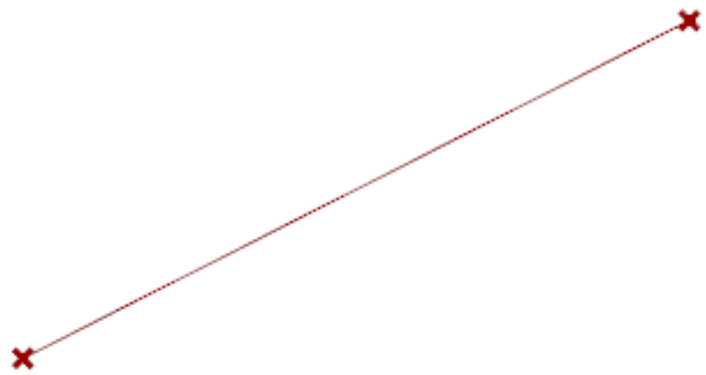


| What you get, is a parametric point whose XYZ position is defined by a slider value. Note that you cannot select this point in Rhino, as it only exists in Grasshopper.

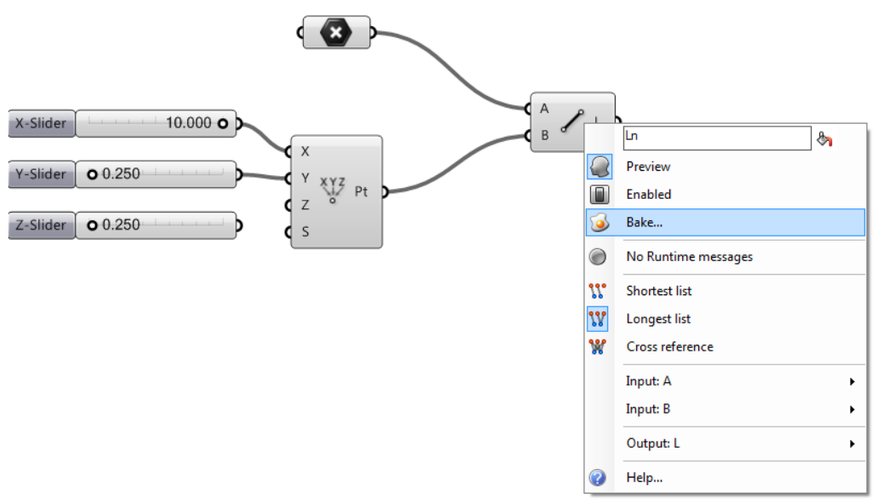
| Next, we'll create a parametric line. Get the Line component from the Curve tab, subsection Primitive and drag it onto the canvas.

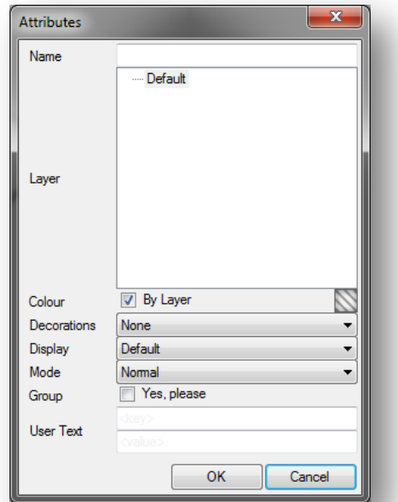
| Connect the referenced point to input A and the slider-defined point to input B. You will receive a parametric line that moves with the points.





| To get the line into Rhinoceros, e.g. to print it or to export it, you have to “Bake” it. Right-click on the line component and click on Bake. Choose a layer and confirm with OK.

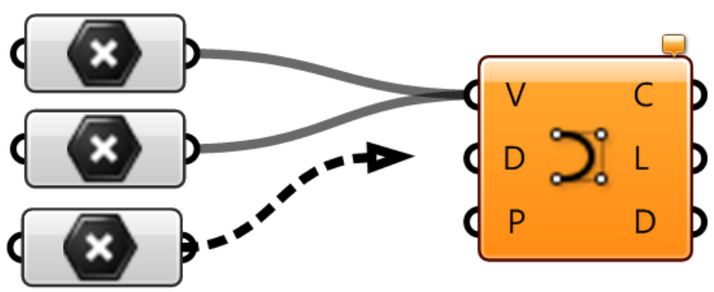




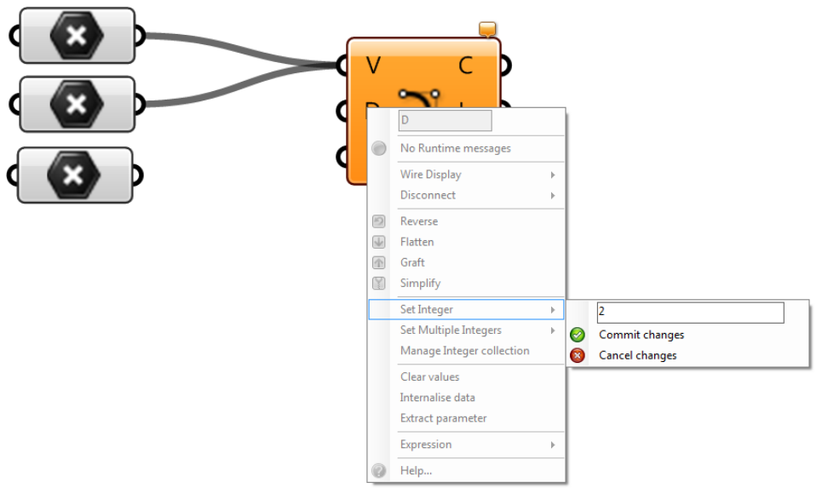
| The line will now show up in Rhinoceros. However, it is decoupled from Grasshopper, i.e. if you change the line in Grasshopper, the Rhino-line will stay the same.

| Let's go back to the very first example, the control point curve. Create three points in Rhino and three point component in GH that reference one point each. Put a Control Curve component from the Curve tab, Spline subsection next to it.

| You can connect multiple components to one input. Keep the Shift key pressed, while dragging the wire from the point component to the V[ertex] input of the curve. The order is important!

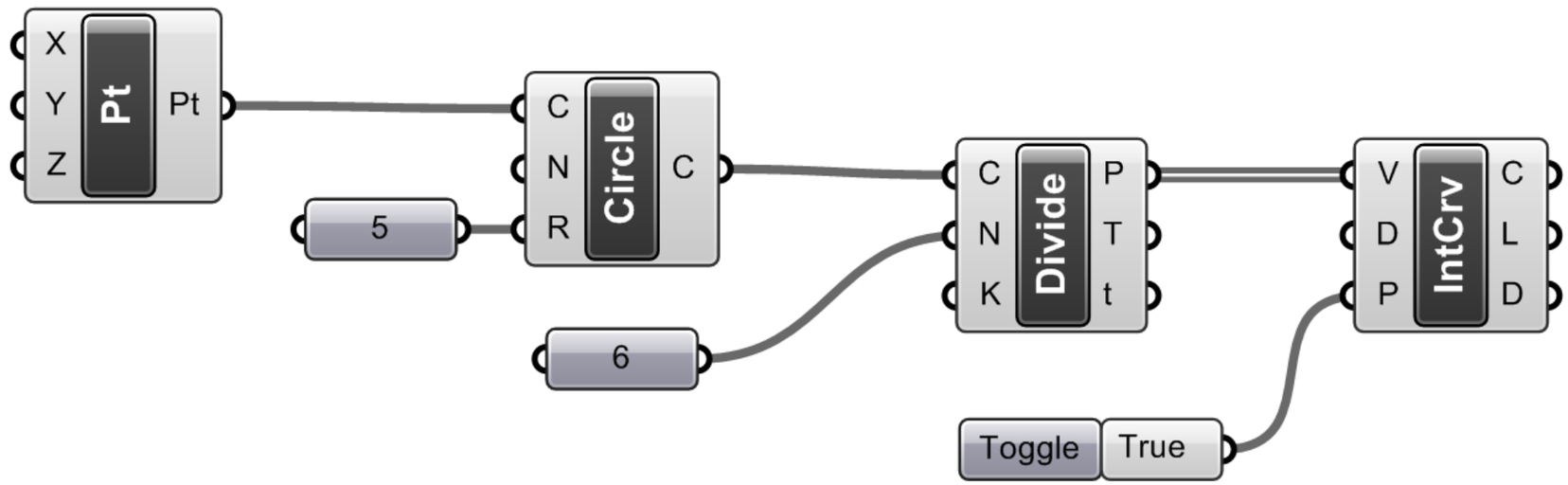


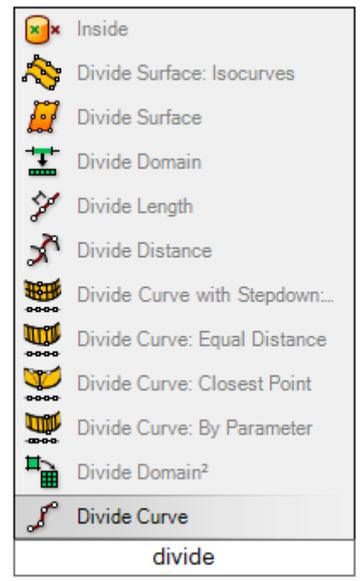
| Even after you connected the points, the curve component is still yellow. When you mouse over, it will tell you that the degree must be less than the number of points (remember Barch geometry!). The D input sets the degree of the curve. We can either connect a number component (even a slider!) to the input, or we can set it inside the component. To do so, right-click the D and go to Set Integer. Enter “2” and confirm by clicking the green check-icon.



| Try to find out what the P input does and how you can change it!

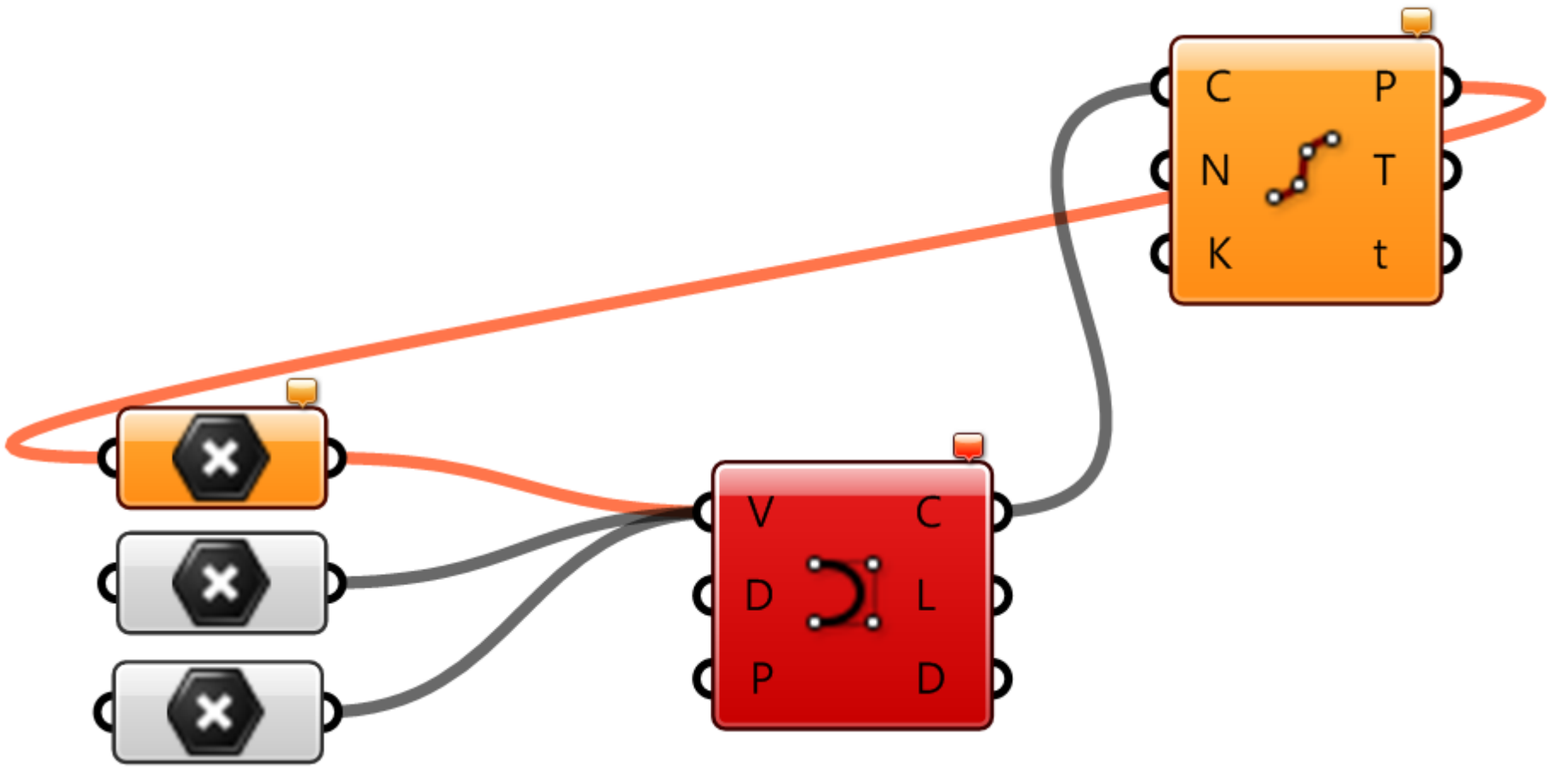
| Can you guess what the following definition creates? If not, try it out.





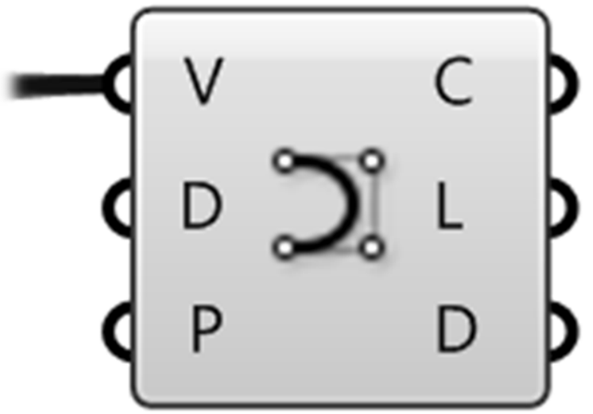
| If you cannot find some of the components, try searching for them. Double-click in empty canvas space and enter the first few letters of the component.

| Two more important aspects: Definitions must never loop!

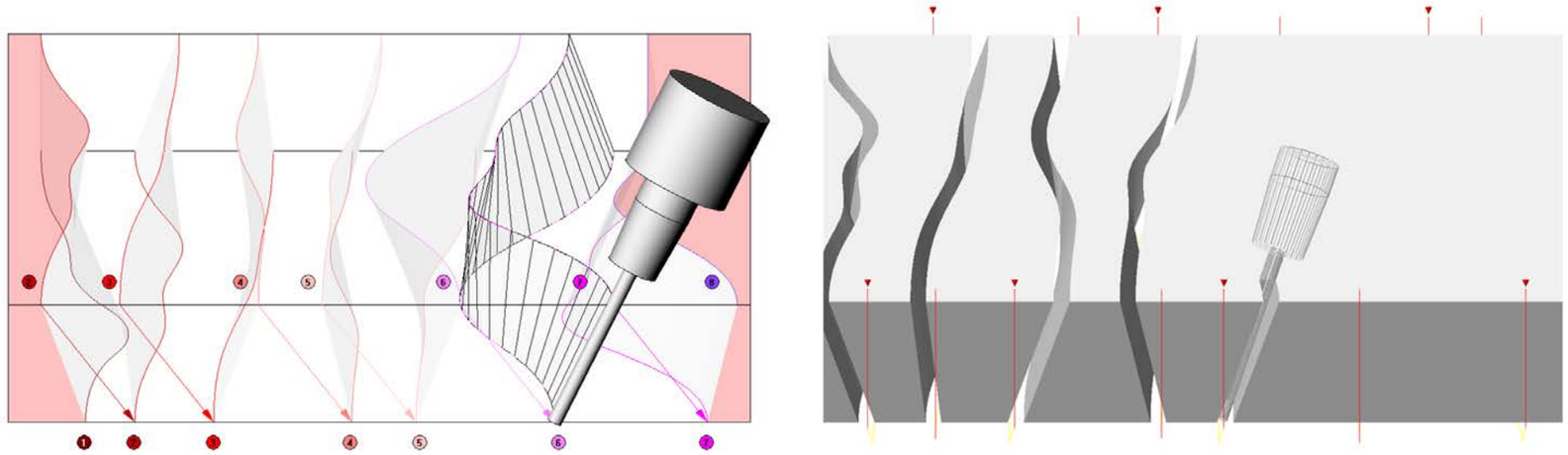


| Every GH component contains code, but the user cannot access it, i.e. it's a black box. Because of this, the user can focus on the logic, instead of having to write code himself.

```
protected override void SolveInstance(IGH_DataAccess DA)
{
  int destination = 0;
  bool flag = false;
  if (DA.GetData<int>(1, ref destination) && DA.GetData<bool>(2, ref flag))
  {
    if (destination < 1)
    {
      this.AddRuntimeMessage(GH_RuntimeMessageLevel.Error, "Degree must be higher than or equal to 1.");
      destination = 1;
    }
    if (destination > 11)
    {
      this.AddRuntimeMessage(GH_RuntimeMessageLevel.Error, "Degree must be smaller than or equal to 11.");
      destination = 11;
    }
    List<GH_Point> list = new List<GH_Point>();
    if (DA.GetDataList<GH_Point>(0, list))
    {
      List<Point3d> points = new List<Point3d>();
      foreach (GH_Point point in list)
      {
        if ((point != null) && point.IsValid)
        {
          points.Add(point.Value);
        }
      }
      if (points.Count < 2)
      {
        this.AddRuntimeMessage(GH_RuntimeMessageLevel.Error, "Insufficient vertices for a curve");
      }
      else
      {
        if (destination >= points.Count)
        {
          this.AddRuntimeMessage(GH_RuntimeMessageLevel.Blank | GH_RuntimeMessageLevel.Warning, "The degree must be less than the number of control-points");
          destination = points.Count - 1;
        }
        NurbsCurve data = NurbsCurve.Create(flag, destination, points);
        if (data != null)
        {
          DA.SetData(0, data);
          double length = data.GetLength();
          if (RhinoMath.IsValidDouble(length))
          {
            DA.SetData(1, length);
          }
          else
          {
            DA.SetData(1, 0.0);
          }
          DA.SetData(2, data.Domain);
        }
      }
    }
  }
}
```

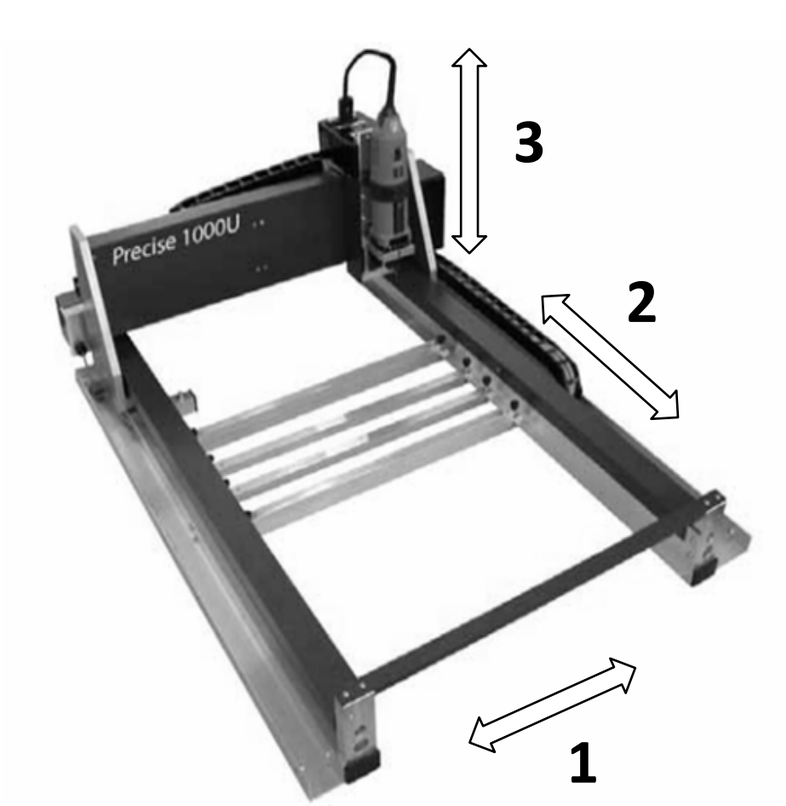


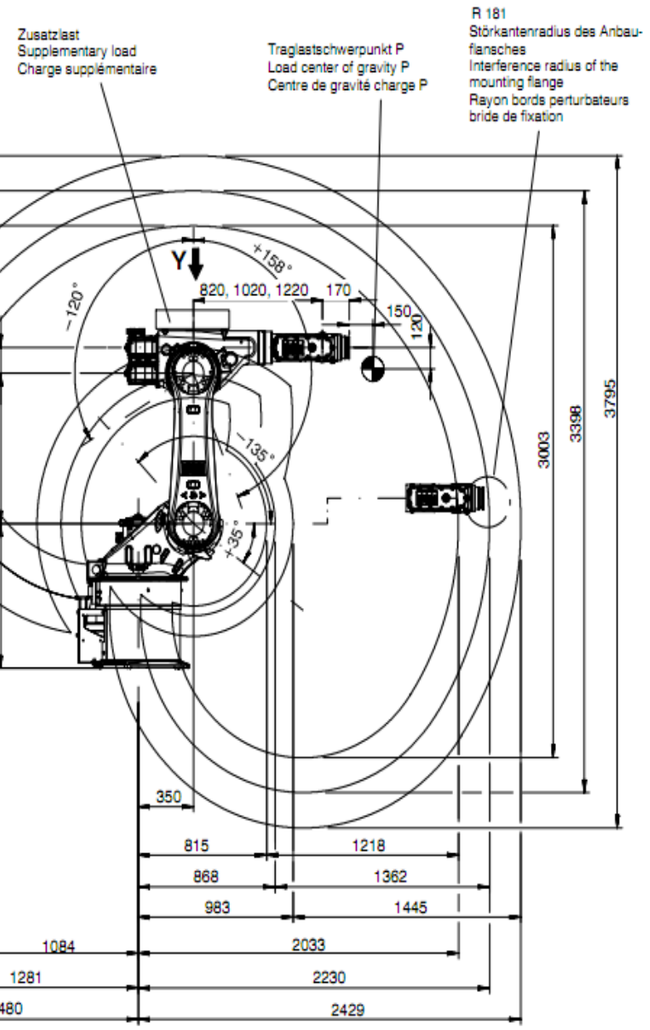
Robots basically work like CNC machines: The first step is generating toolpaths, and the second is translating that data into a format that the machine can understand.



```
&ACCESS RVP
&REL 1
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM EDITMASK = *
DEFDAT kukaprc_project
;FOLD EXTERNAL DECLARATIONS;%{PE}%MKUKATPBASIS,%CEXT,%VCOMMON,%P
;FOLD BASISTECH EXT;%{PE}%MKUKATPBASIS,%CEXT,%VEXT,%P
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL INT SUCCESS
;ENDFOLD (BASISTECH EXT)
;FOLD USER EXT;%{E}%MKUKATPUSER,%CEXT,%VEXT,%P
;Make here your modifications
;ENDFOLD (USER EXT)
;ENDFOLD (EXTERNAL DECLERATIONS)
INT CR_GENNUMBER = 525475
INT CR_ANSWER
DECL PDAT PPDATP2={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP2={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL E6AXIS XP2 = {A1 5,A2 -90,A3 100,A4 5,A5 -10,A6 -5,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}
DECL PDAT PPDATP4={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP4={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL FRAME XP4={X 408.930, Y 408.930, Z 118.896, A 45.000, B 16.576, C 179.321}
DECL PDAT PPDATP5={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP5={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL FRAME XP5={X 532.239, Y 554.277, Z 755.706, A 41.949, B 16.576, C 177.653}
DECL PDAT PPDATP3={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP3={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL E6AXIS XP3 = {A1 5,A2 -90,A3 100,A4 5,A5 -10,A6 -5,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}
ENDDAT
```

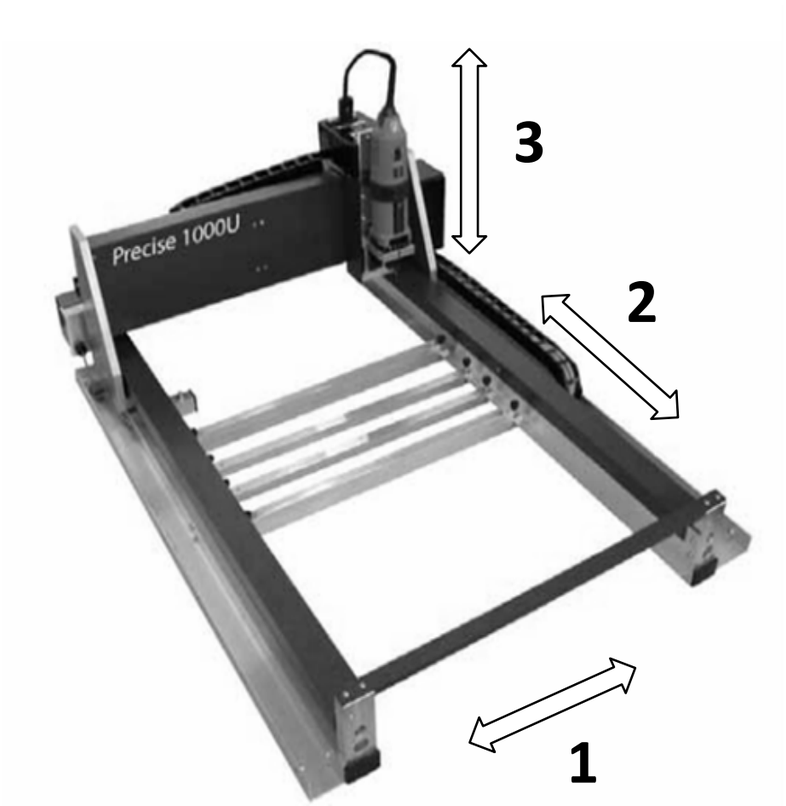
| We'll start with the second step. Try to figure out what control data a three axis milling machine (left) requires, as opposed to a six-axis industrial robot.

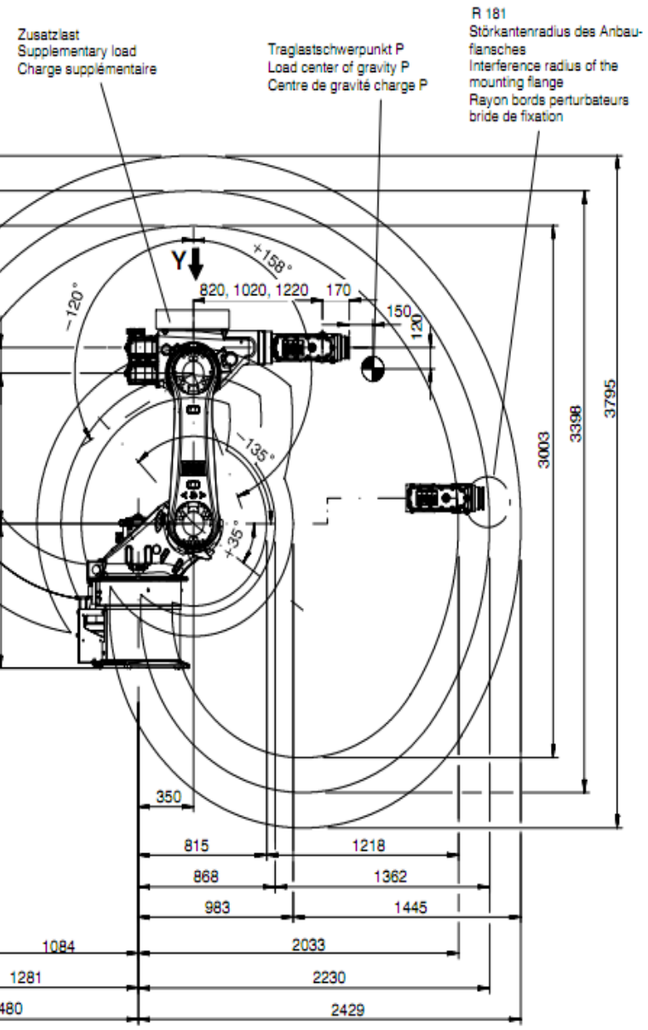




| We'll start with the second step. Try to figure out what control data a three axis milling machine (left) requires, as opposed to a six-axis industrial robot.

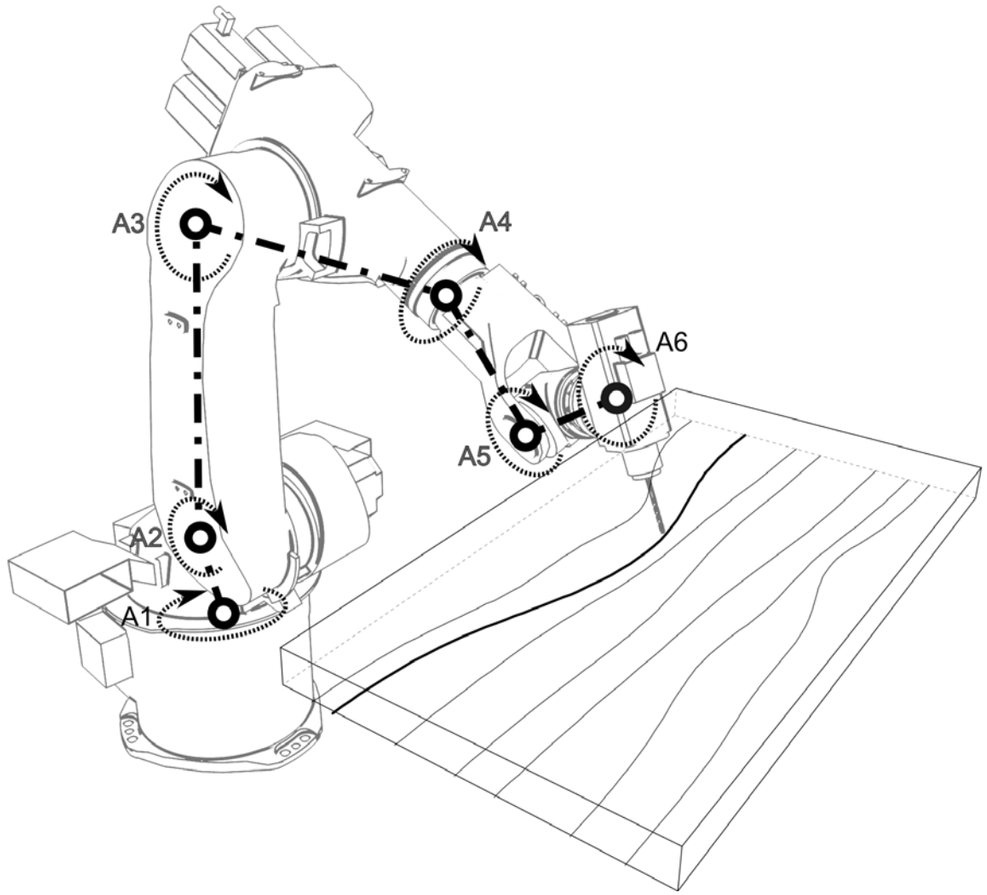
XYZ



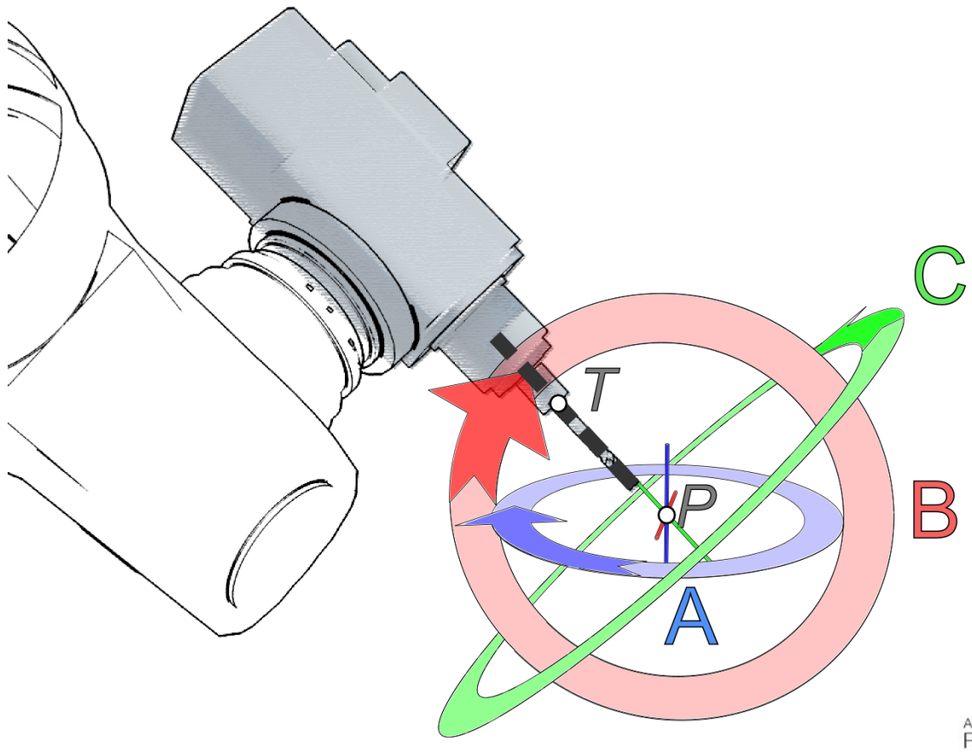


| A six axis robot either requires one rotation value for each axis, or a defined toolframe via XYZ ABC. This creates a plane, defining the position of the end-effector in 3D space.

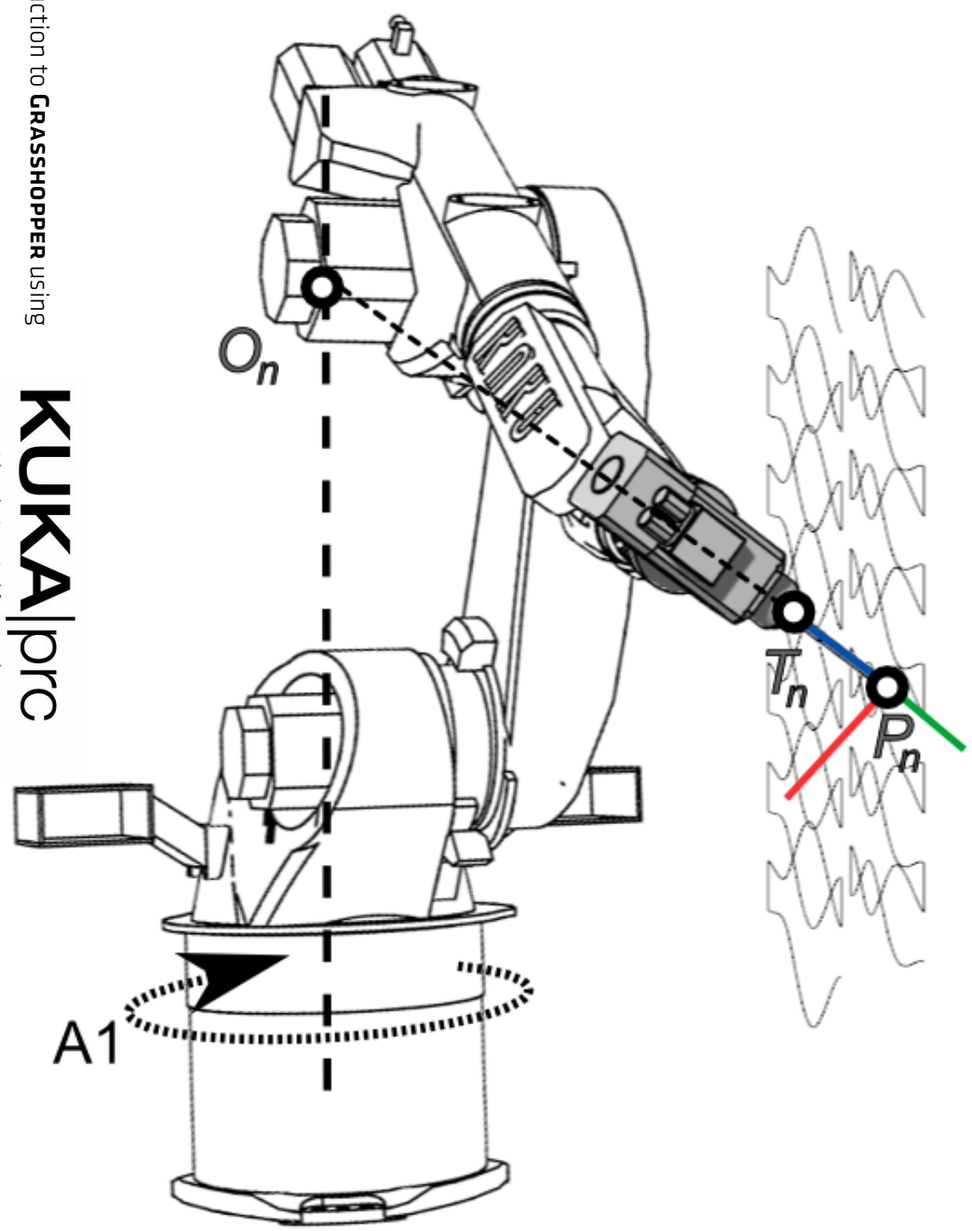
A01 A02 A03 A04 A05 A06

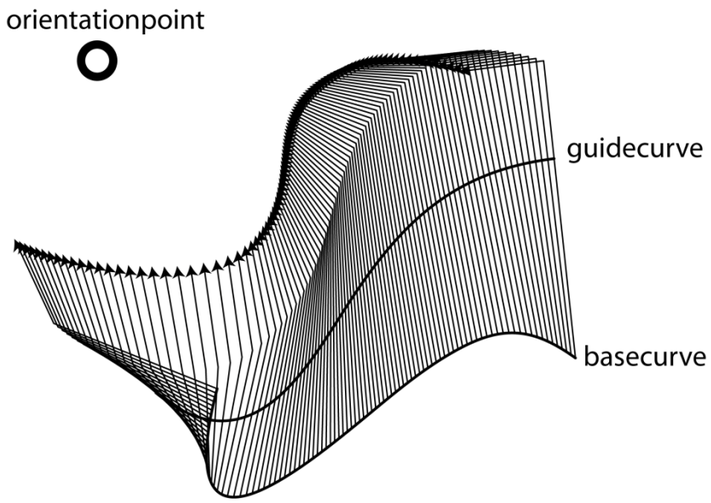


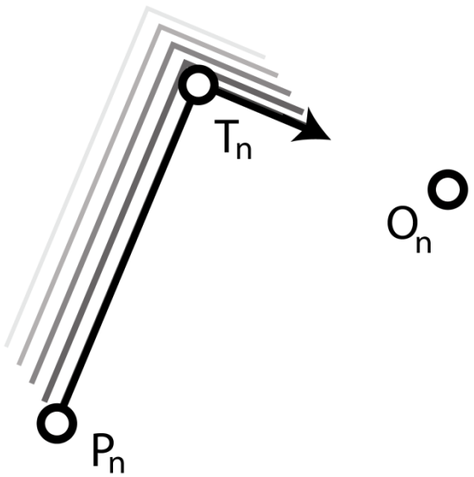
XYZ ABC



| Our task is therefore to create toolpaths that contain enough information to control a robot, i.e. we need three points at every toolpath position to define the toolplane

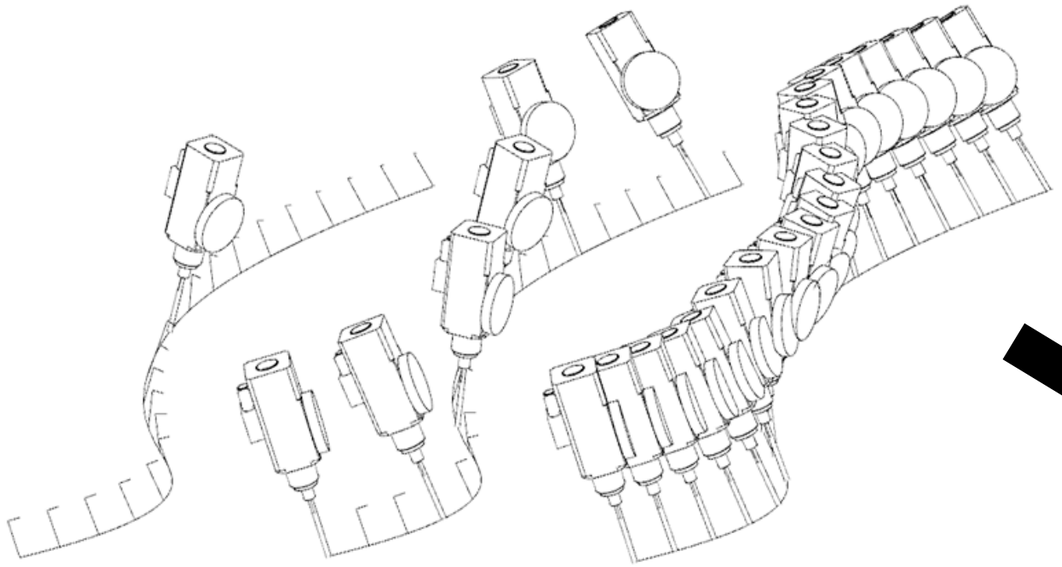






Association for
Robots in Architecture

| Once we have the toolpaths, KUKA|prc will take care of the formatting and code.

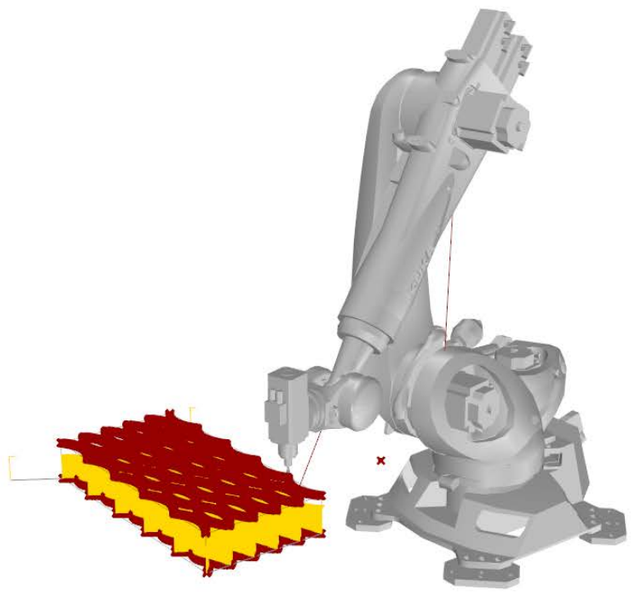


```
&ACCESS RVP
&REL 1
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM EDITMASK = *
DEFDAT kukaprc_project
;FOLD EXTERNAL DECLARATIONS;%{PE}%MKUKATPBASIS,%CEXT,%VCOMMON,%P
;FOLD BASISTECH EXT;%{PE}%MKUKATPBASIS,%CEXT,%VEXT,%P
EXT  BAS (BAS_COMMAND  :IN,REAL  :IN )
DECL INT SUCCESS
;ENDFOLD (BASISTECH EXT)
;FOLD USER EXT;%{E}%MKUKATPUSER,%CEXT,%VEXT,%P
;Make here your modifications
;ENDFOLD (USER EXT)
;ENDFOLD (EXTERNAL DECLERATIONS)
INT CR_GENNUMBER = 525475
INT CR_ANSWER
DECL PDAT PPDATP2={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP2={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL E6AXIS XP2 = {A1 5,A2 -90,A3 100,A4 5,A5 -10,A6 -5,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}
DECL PDAT PPDATP4={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP4={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL FRAME XP4={X 408.930, Y 408.930, Z 118.896, A 45.000, B 16.576, C 179.321}
DECL PDAT PPDATP5={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP5={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL FRAME XP5={X 532.239, Y 554.277, Z 755.706, A 41.949, B 16.576, C 177.653}
DECL PDAT PPDATP3={VEL 50,ACC 100,APO_DIST 100}
DECL FDAT FP3={TOOL_NO 0,BASE_NO 0,IPO_FRAME #BASE,POINT2[] " "}
DECL E6AXIS XP3 = {A1 5,A2 -90,A3 100,A4 5,A5 -10,A6 -5,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}
ENDDAT
```

| Now, we'll go from toolpath design to robotic fabrication

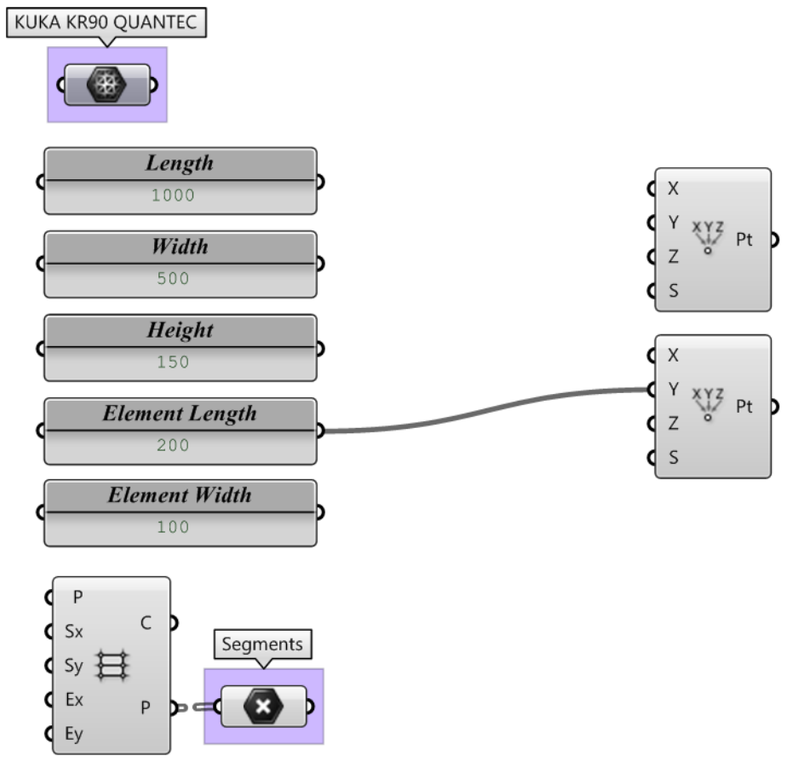
| Let's start by opening the provided Rhinoceros file. The unit of choice for fabrication is millimetres, therefore this file is in mm as well. This file contains a 3D model of our industrial robot as well as the outline of our stock model

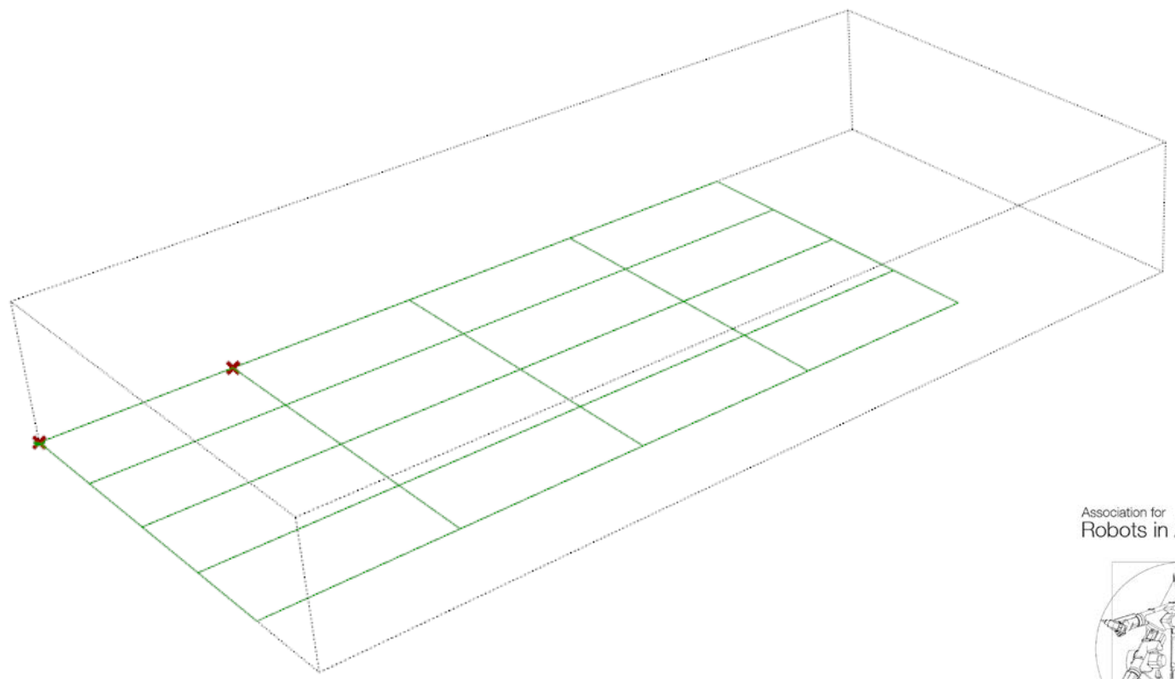
| Next launch Grasshopper and open the provided file. It contains several key numbers for our project, as well as a reference to the 3D robot model.



| We'll start out with creating just a few curves, which are then distributed over the stock model, but still controllable via sliders. These will be our toolpaths!

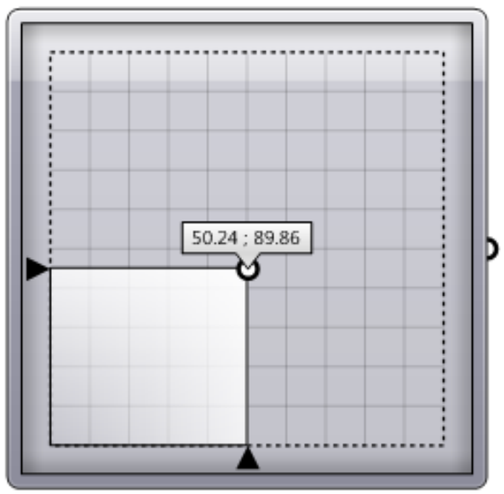
| Our initial curve should be as long as one segment and consist of three points: A fixed start and end point, with a variable point in the middle. Put 2 Point XYZ components onto the canvas, leave the first one at the default value of 0,0,0. For the second Point XYZ component, plug the "Element Length" value into its Y-value.

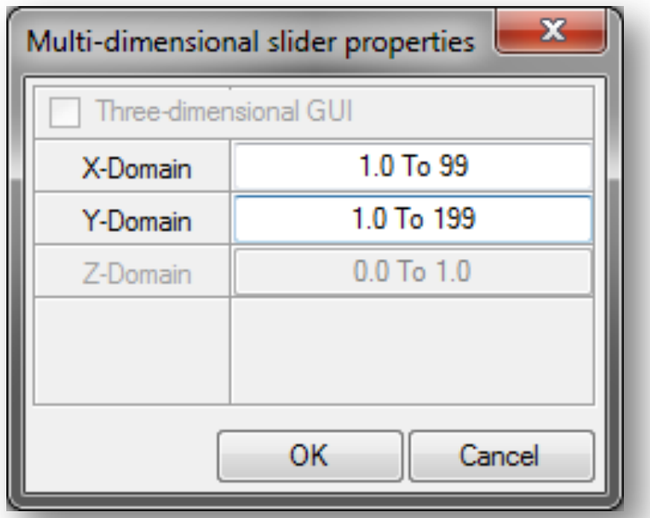




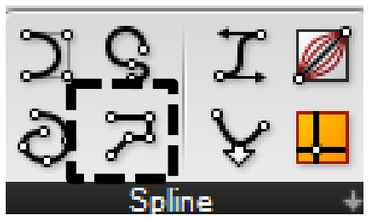
| For the flexible middle point, we'll do something special. Get a so-called MD-Slider from the Params tab, Special subsection.

| Double-click the slider on the canvas, you can now set the range of numbers, that the slider will generate. Set the X-domain to 1.0-99.0 and the Y-domain to 1.0-199.0. This will ensure that the middle point will not overlap with the two other points.

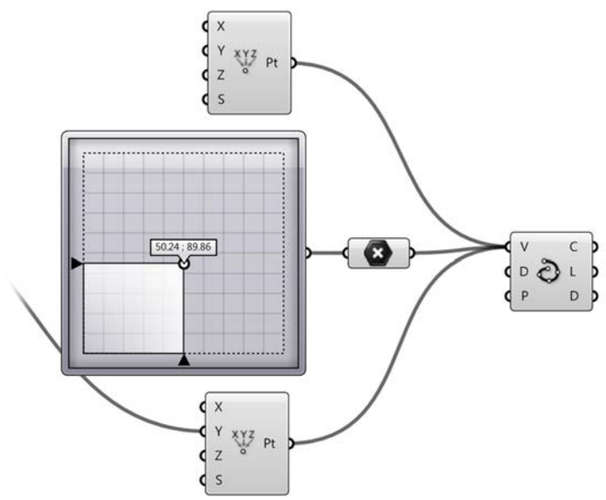


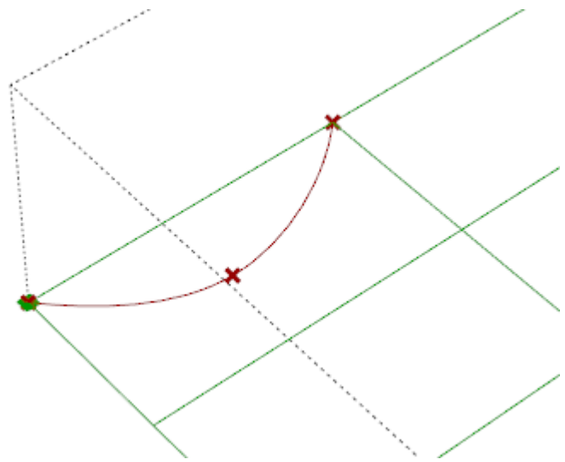


| Connect the output of the slider to a point component from Params, subsection Geometry. This converts the numbers into a point. Now put an Interpolate Curve from the Curve tab, Spline subsection onto the canvas.

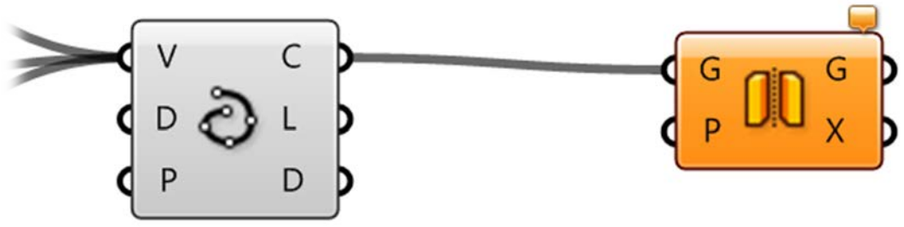


| Keep Shift pressed and connect the three points to the V[ertex] input of the component. First the 0,0,0 point, then the flexible middle point and then the 0,200,0 end point.

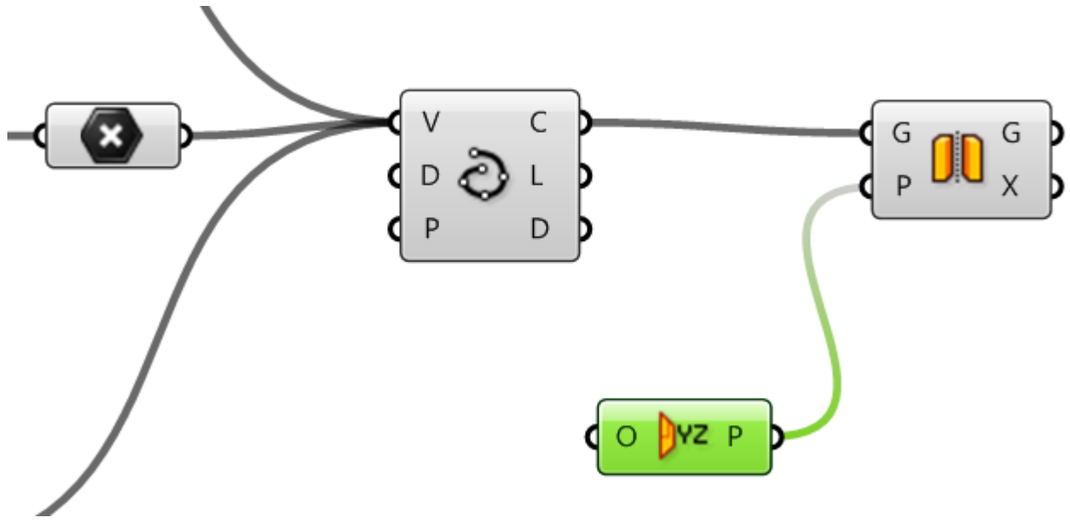




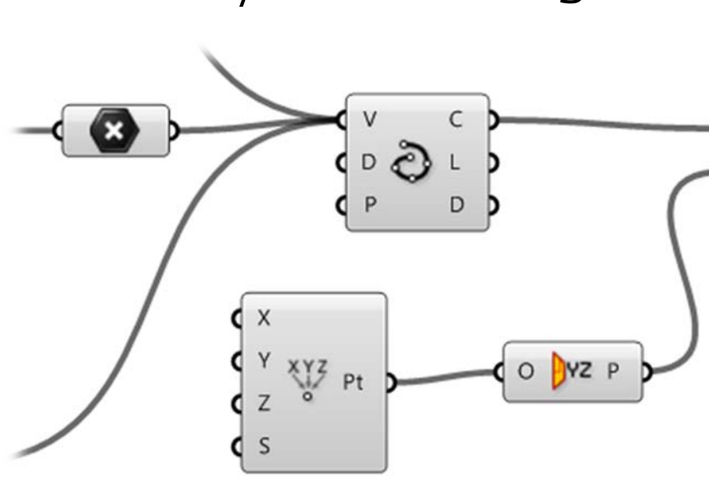
| Now we mirror the curves in the middle of the segment. Get a Mirror component from the Transform tab, Euclidean subsection.

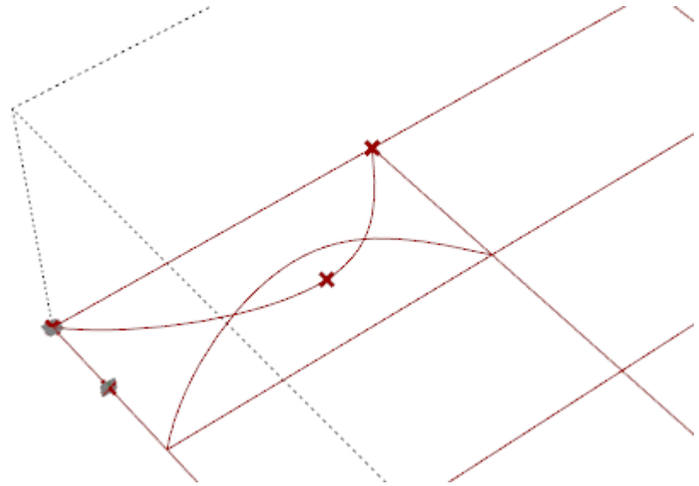


| Connect the curve to the G[eometry] input. You now need to define a mirror plane. As you'll want to mirror at the YZ plane, get the component from the Vector tab, Plane subsection. The origin of the plane can be set by connecting a point to its O[rigin] input.

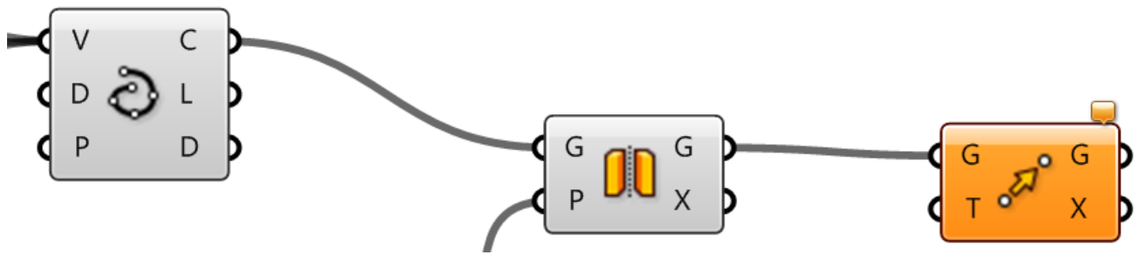


| This plane will have to be in the middle of the segment. Get another Point XYZ component. The segment has a width of 100mm, therefore the X coordinate of the point has to be 100 divided by 2. Set X to 50.

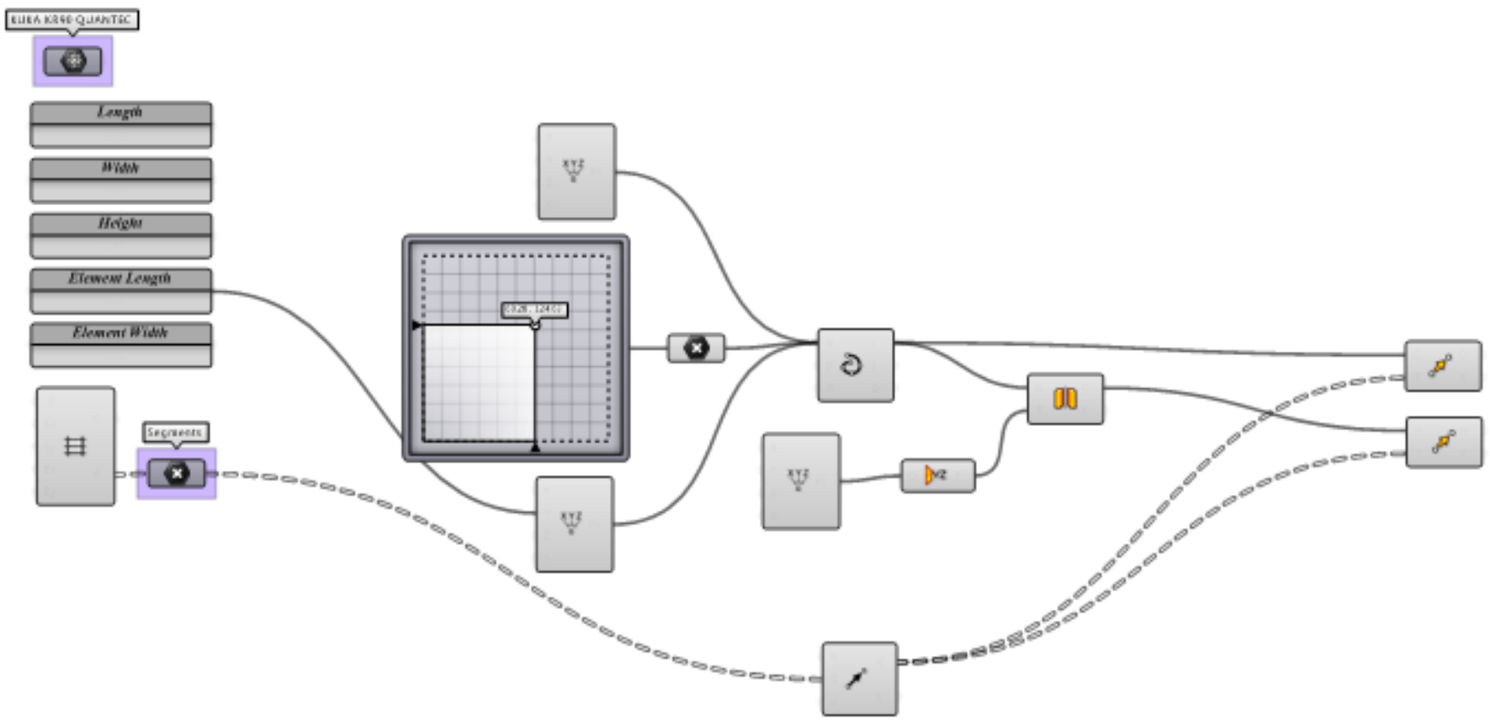


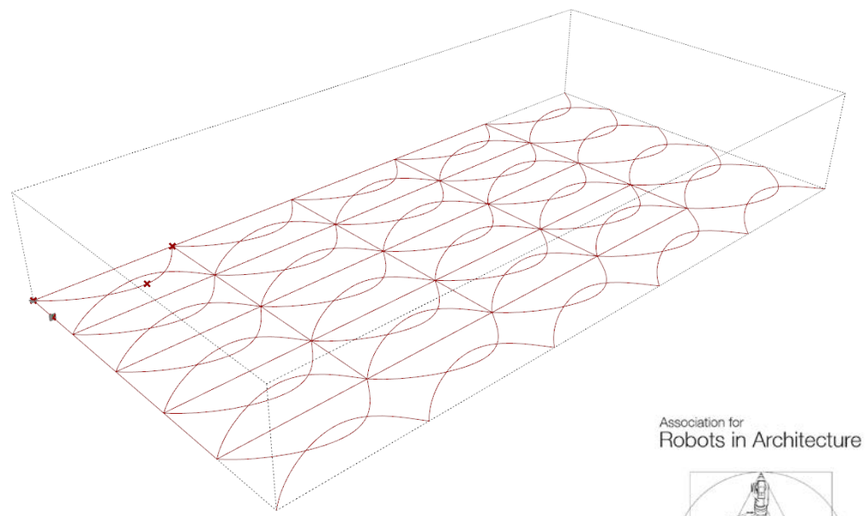


| Now we have to distribute the curves over the whole stockmodel. This can be done in Grasshopper using just one transformation with one start point and many endpoints. Get the Move component from the Transform tab, Euclidean subsection. It has a geometry input and a vector input. Connect the first curve to the G[eometry].

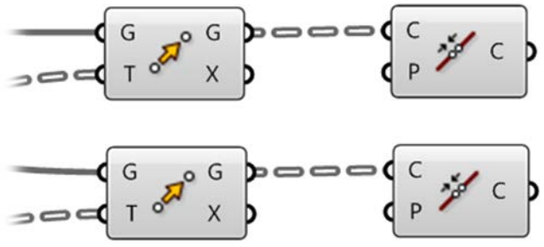


| Get a Vector 2Pt component from the Vector tab, Vector subsection. The A and B values are important. Right-click on A and select Set one Point. To place it at 0,0,0 just enter “0” in the Rhino command prompt and press enter. Connect the list of points coming out of the grid to the B input. This creates 25 vectors for 25 segments. Copy the Move component via Edit/Copy – Edit/Paste and connect the other curve to the duplicated component.

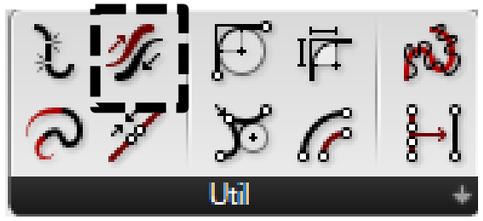


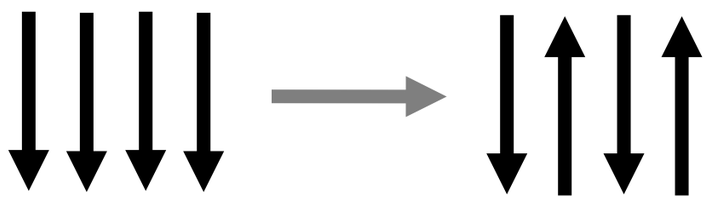


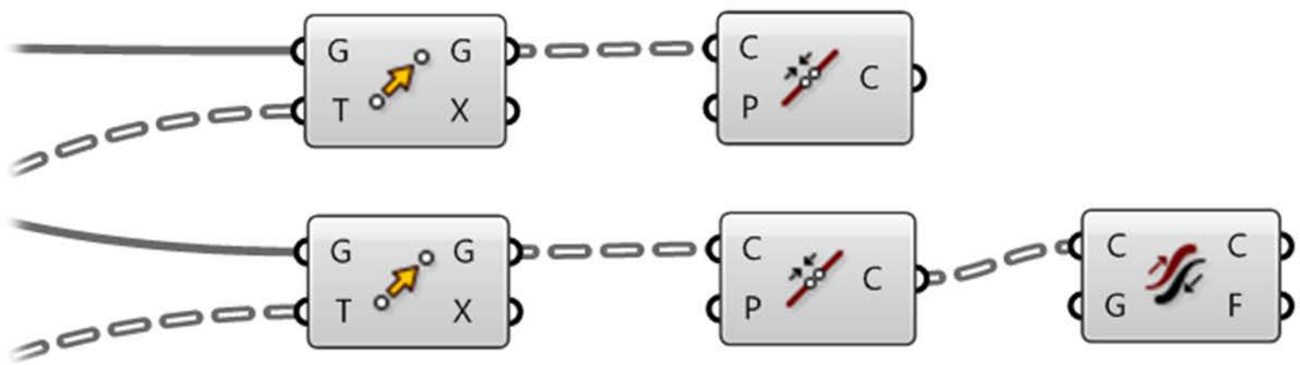
| To create just five curves, out of the 25 segments, get the Join Curves component from the Curve tab, Util subsection. Connect it to both transformed curve lists.



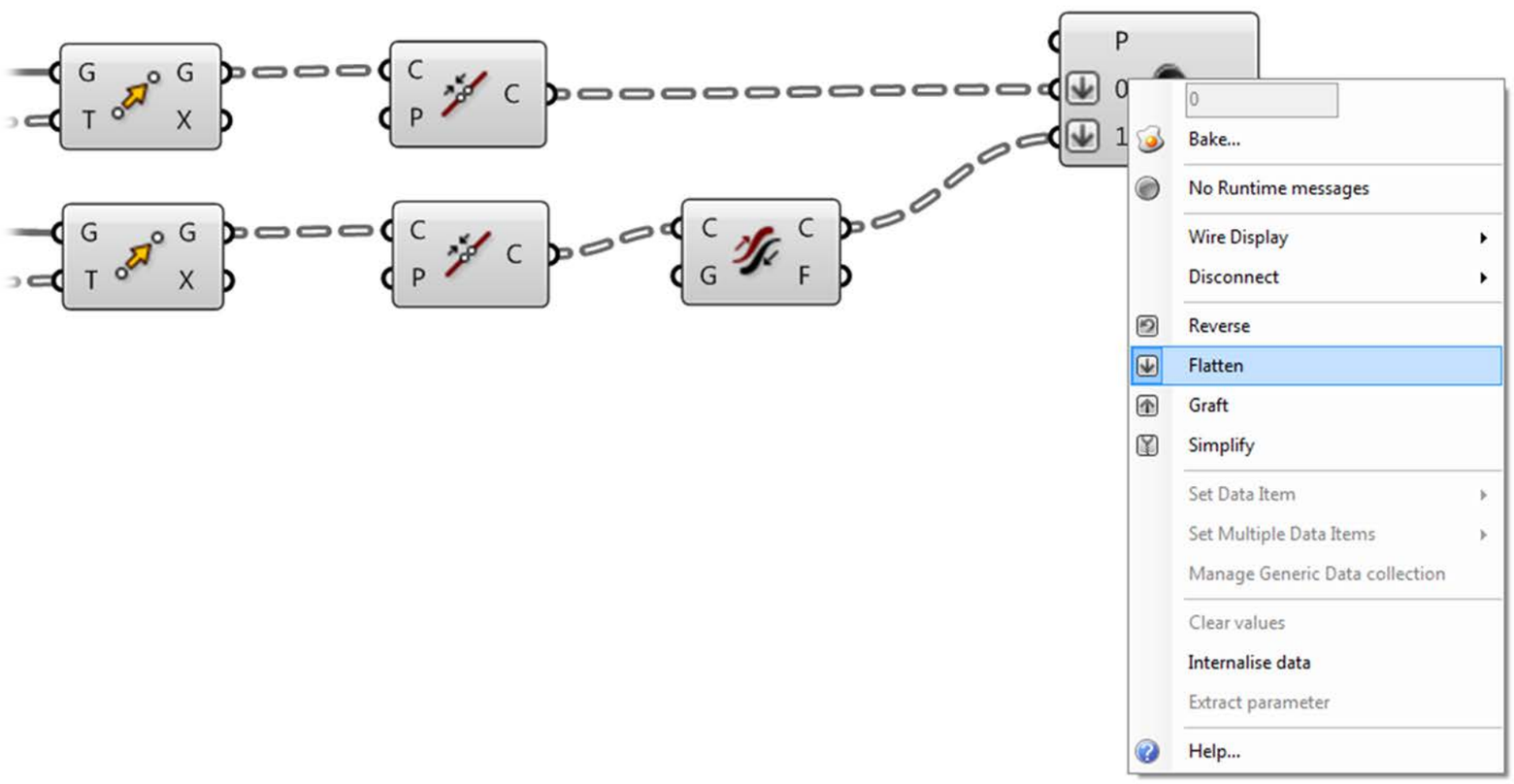
| Right now, all curves are going in the same direction. To get effective movements, we'll have to flip the direction of every second curve. To do so, get the Flip component from the same place. Connect only one list of curves to it.



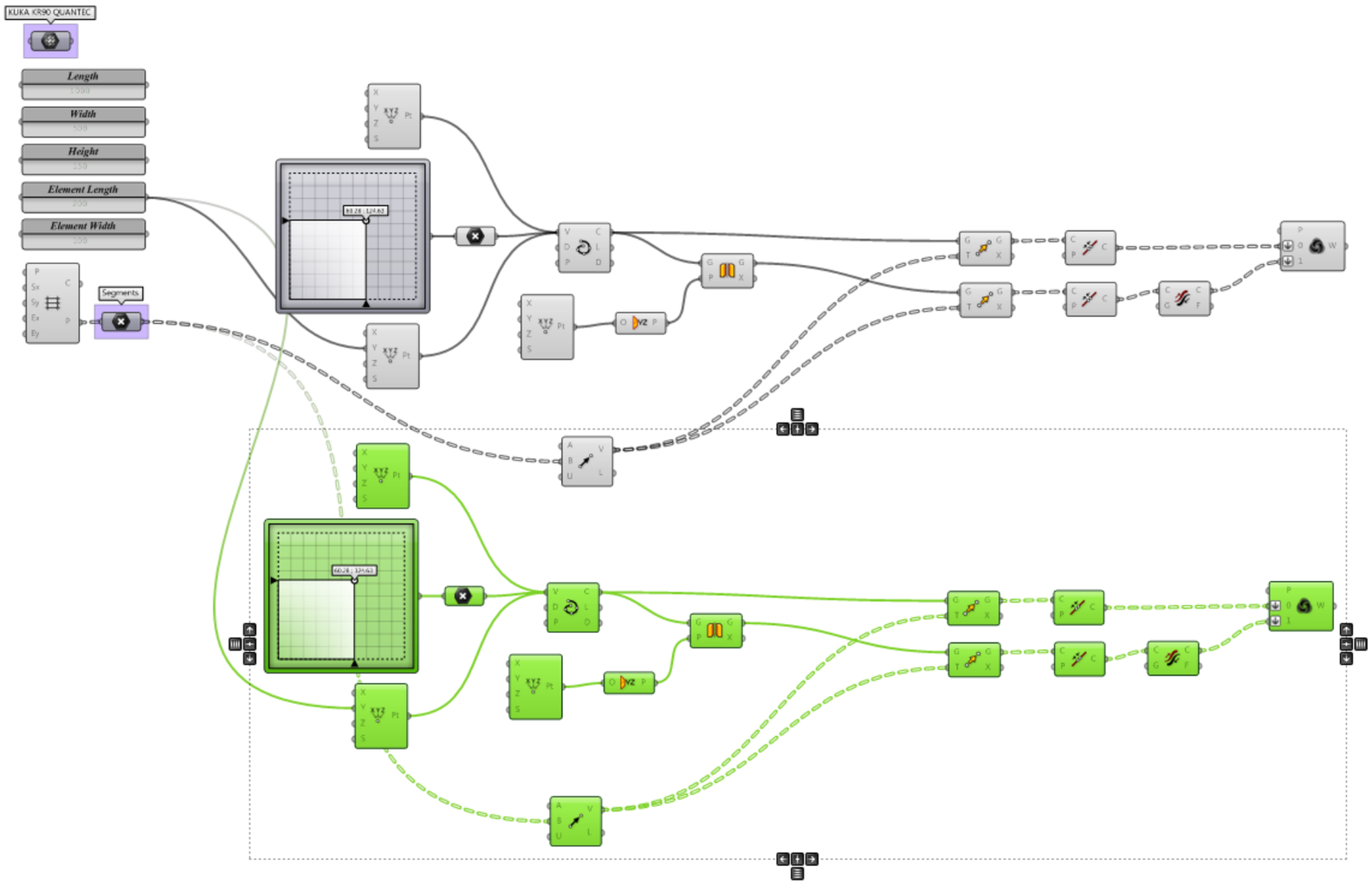




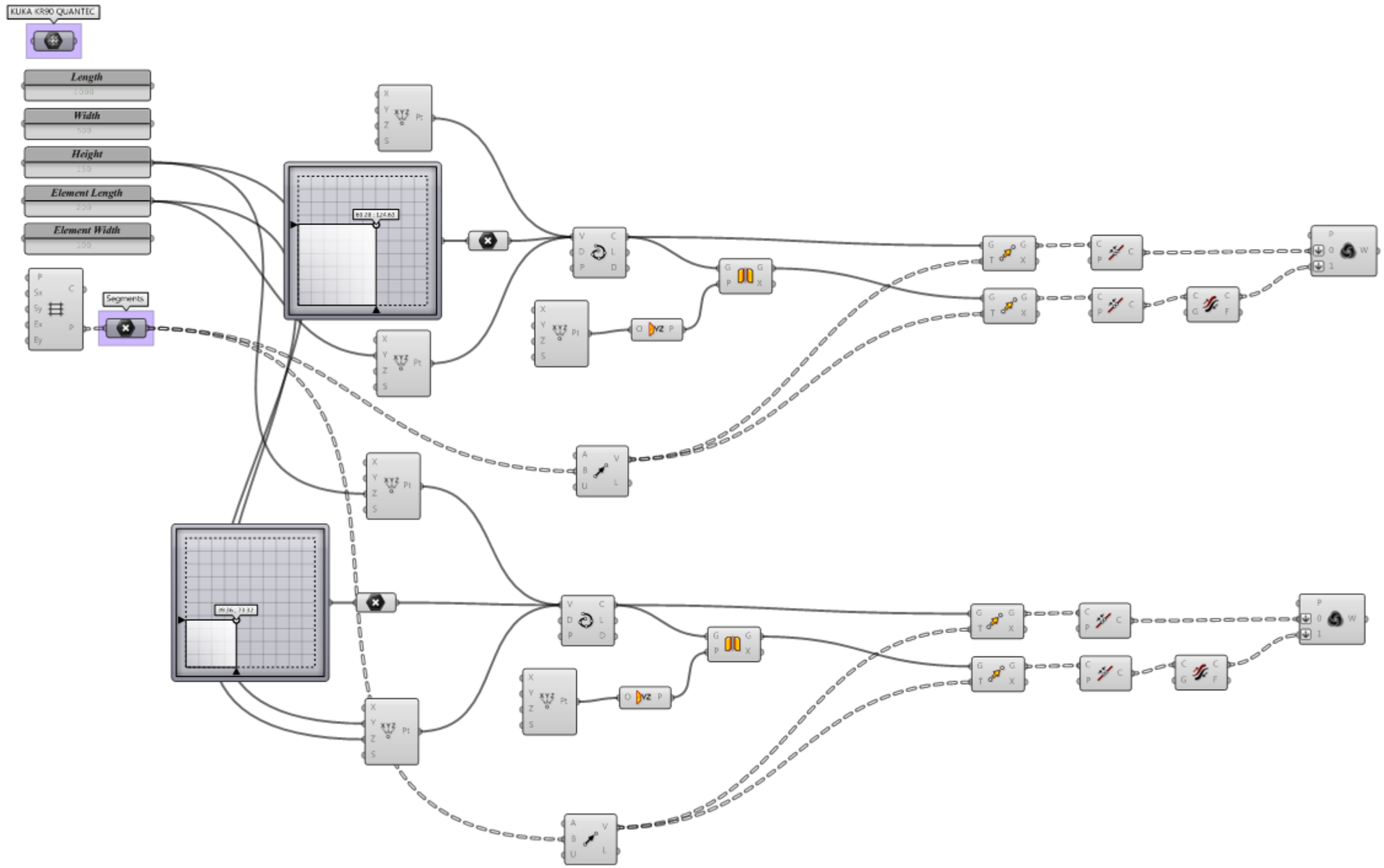
| Now we want to put them back together into one list. Get the Weave component from the Sets tab, List subsection. Connect one list of curves to 0, and the other one to 1. There is one more thing you have to do. Right-click on the input and enable Flatten. Repeat for the other input.



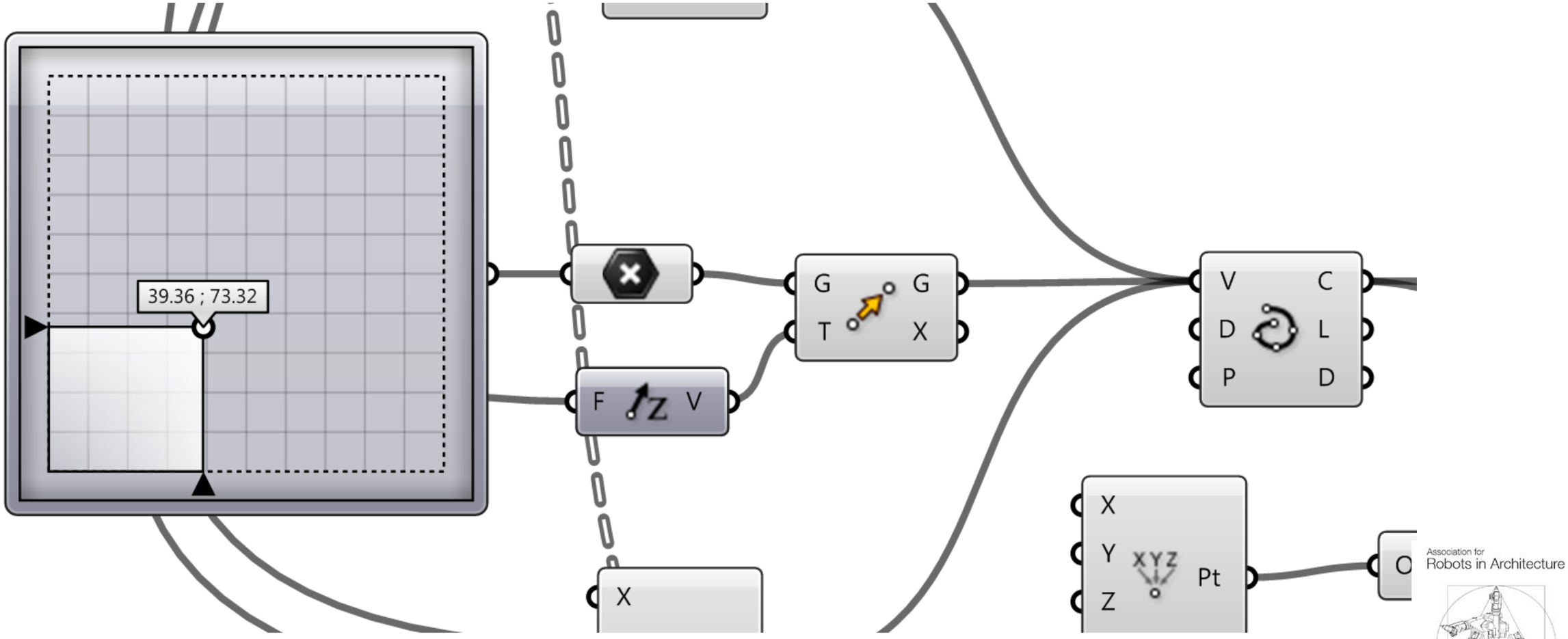
| To create the curves on top of the stockmodel, select all components and duplicate them via copy/paste.



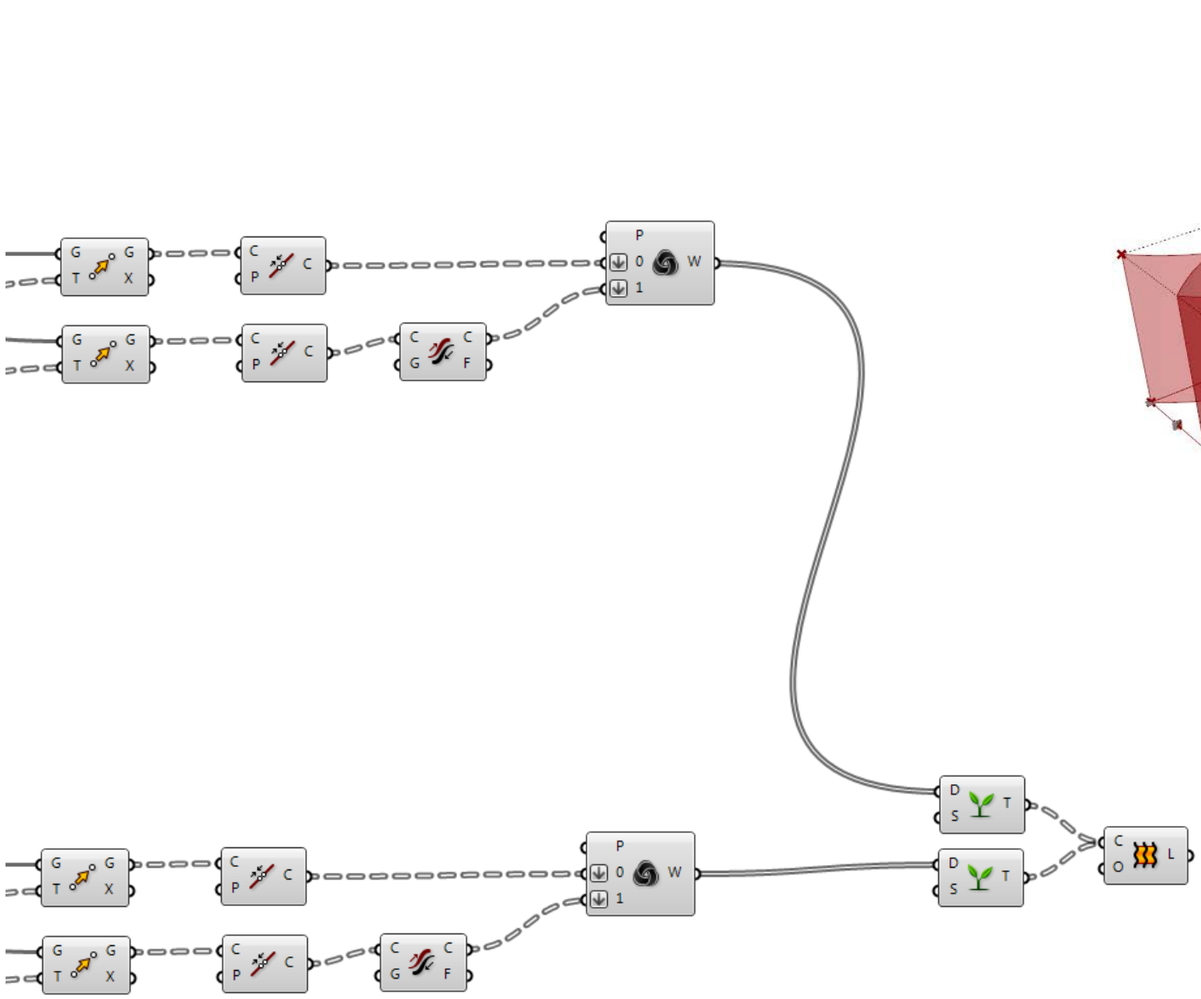
| Now, in order to move them to the top, connect the stockmodel height to the Z-input of all Point XYZ components.

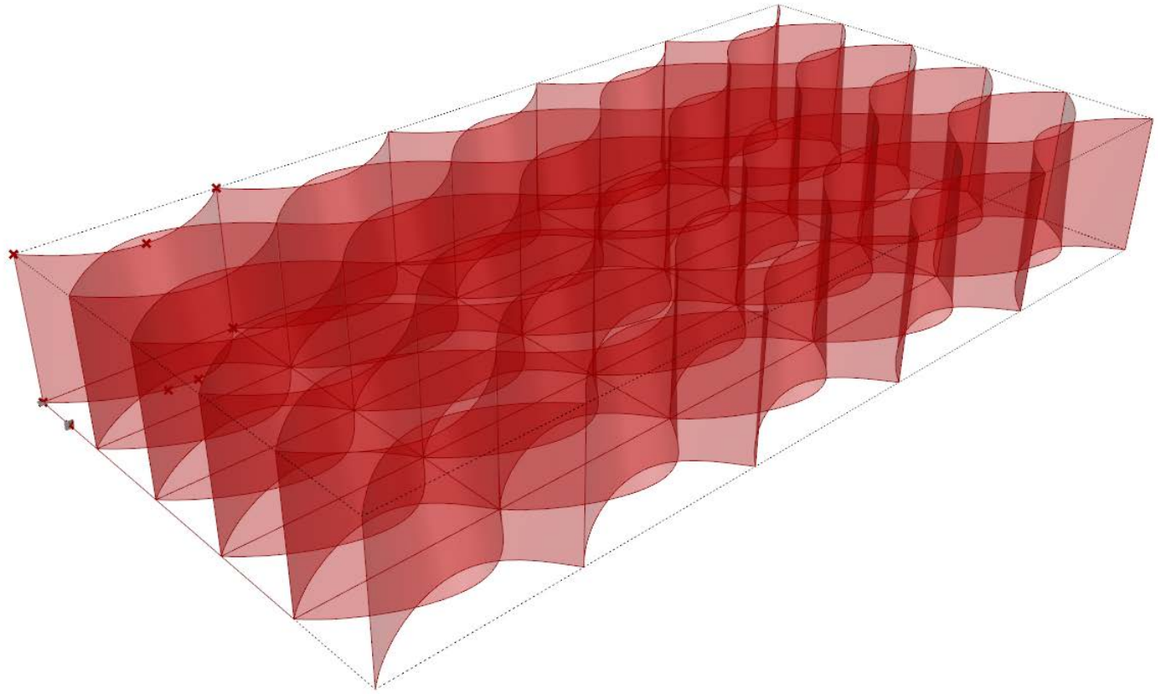


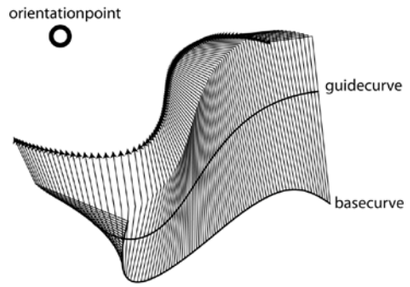
| The only point with you have to move manually is the middle point. Get a Move component from Transform/Euclidean and connect the point to its G[eometry] input. Now add a Unit-Z component from Vector/Vector and connect the height to its F input. The point will now be moved up by 150 units. Re-connect the points in the correct order as before.



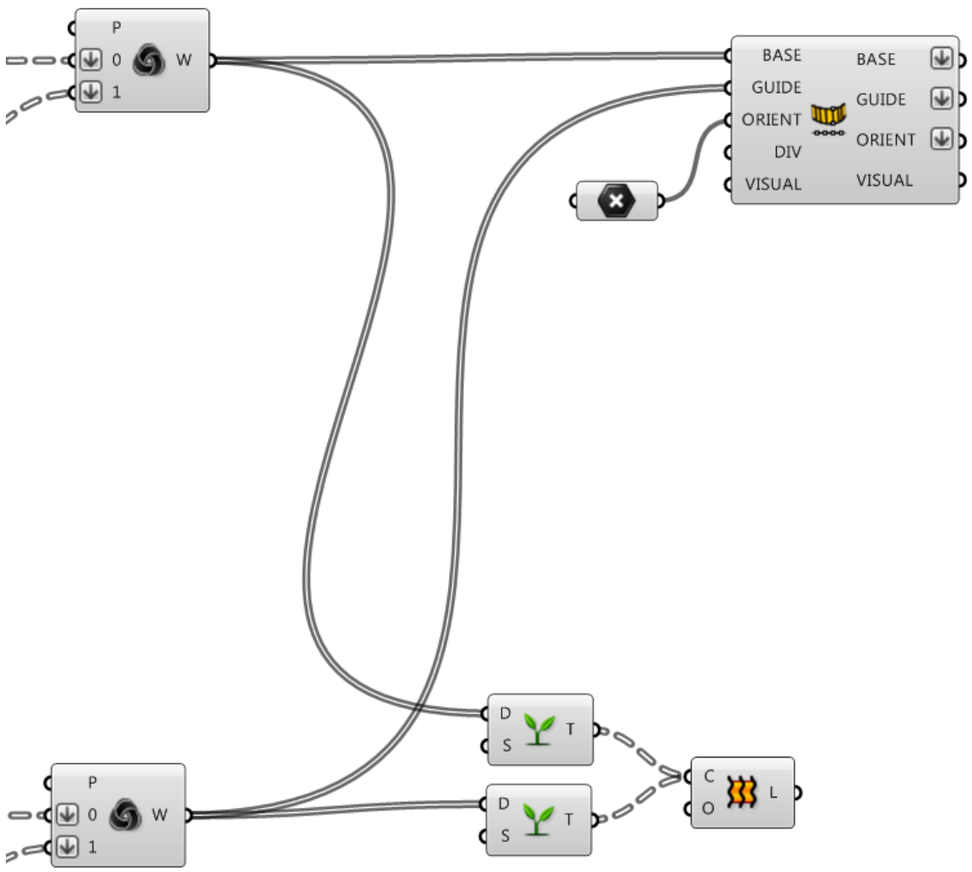
| You can visualize the tool movements by connecting the curves from the Weave output first to a Graft Tree component (from Sets/Tree) and then to a shared Loft component from Surface/Freeform.

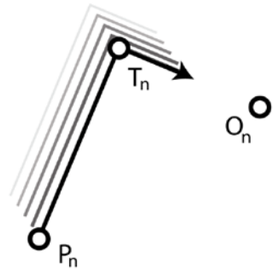


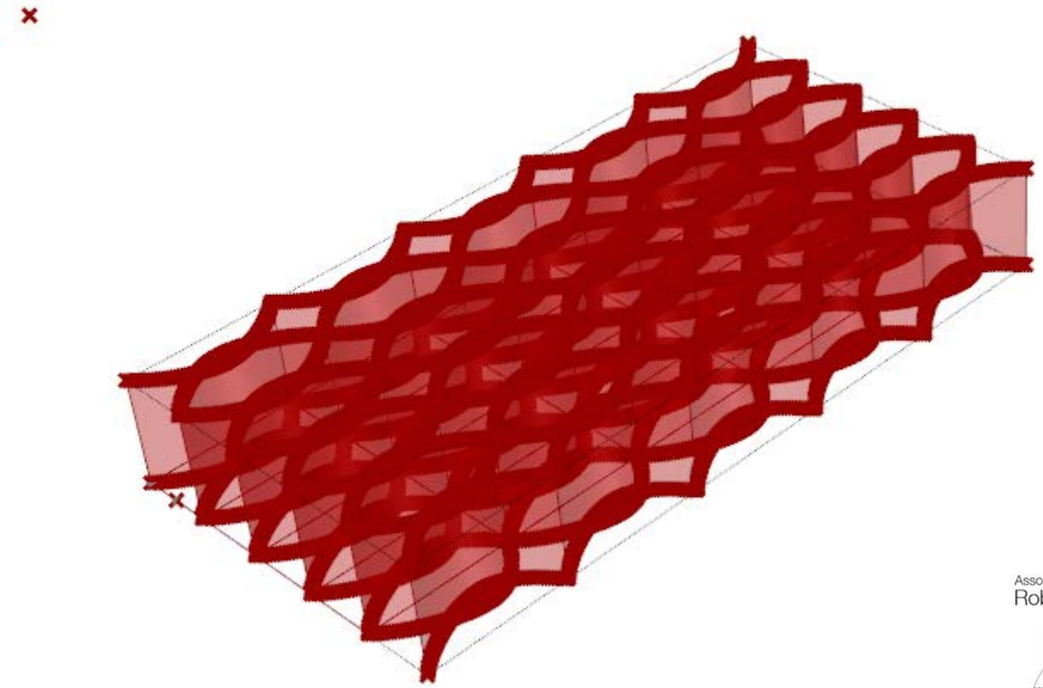




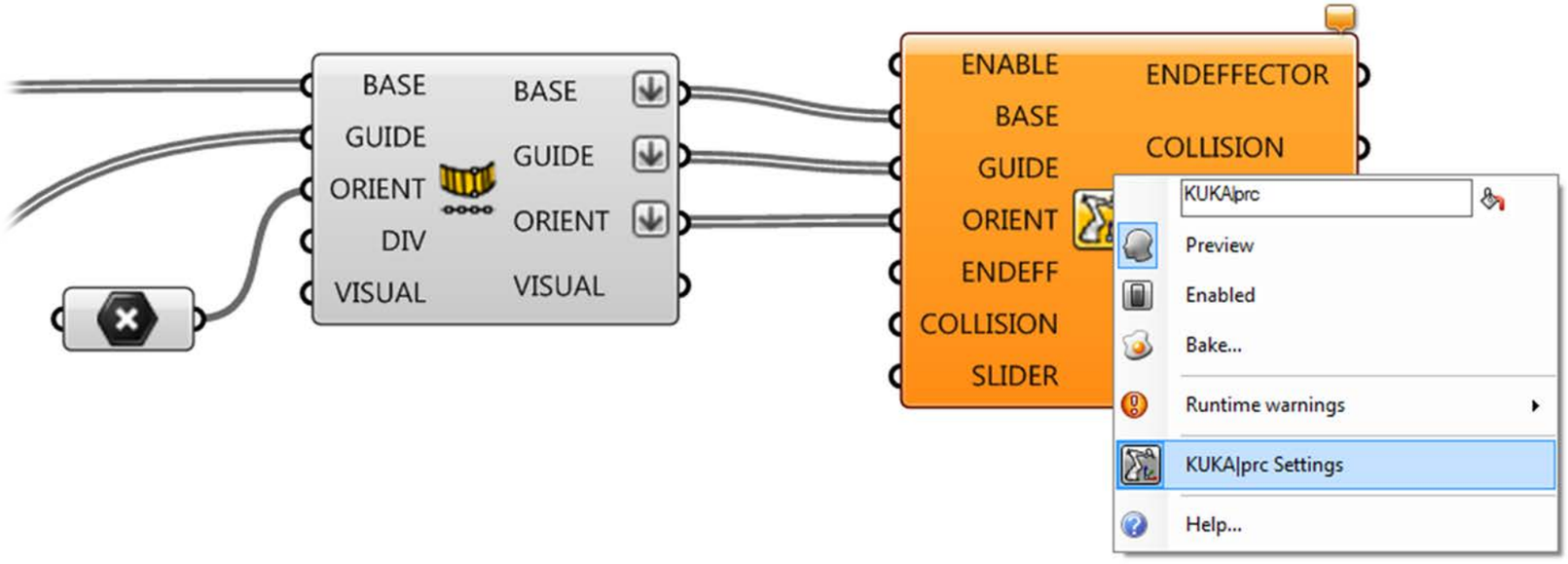
| For the next step, we'll generate the robot toolpaths. From the KUKA|prc tab, get the Divide Curve: Equal Distance component. Connect the lower curves to the base input, and the upper curves to the guide input. For the orient input, get a point from Params and put it somewhere around (-500,0,0). You can set the number if divisions via the Div input. Enable Flatten for all outputs via right-click.



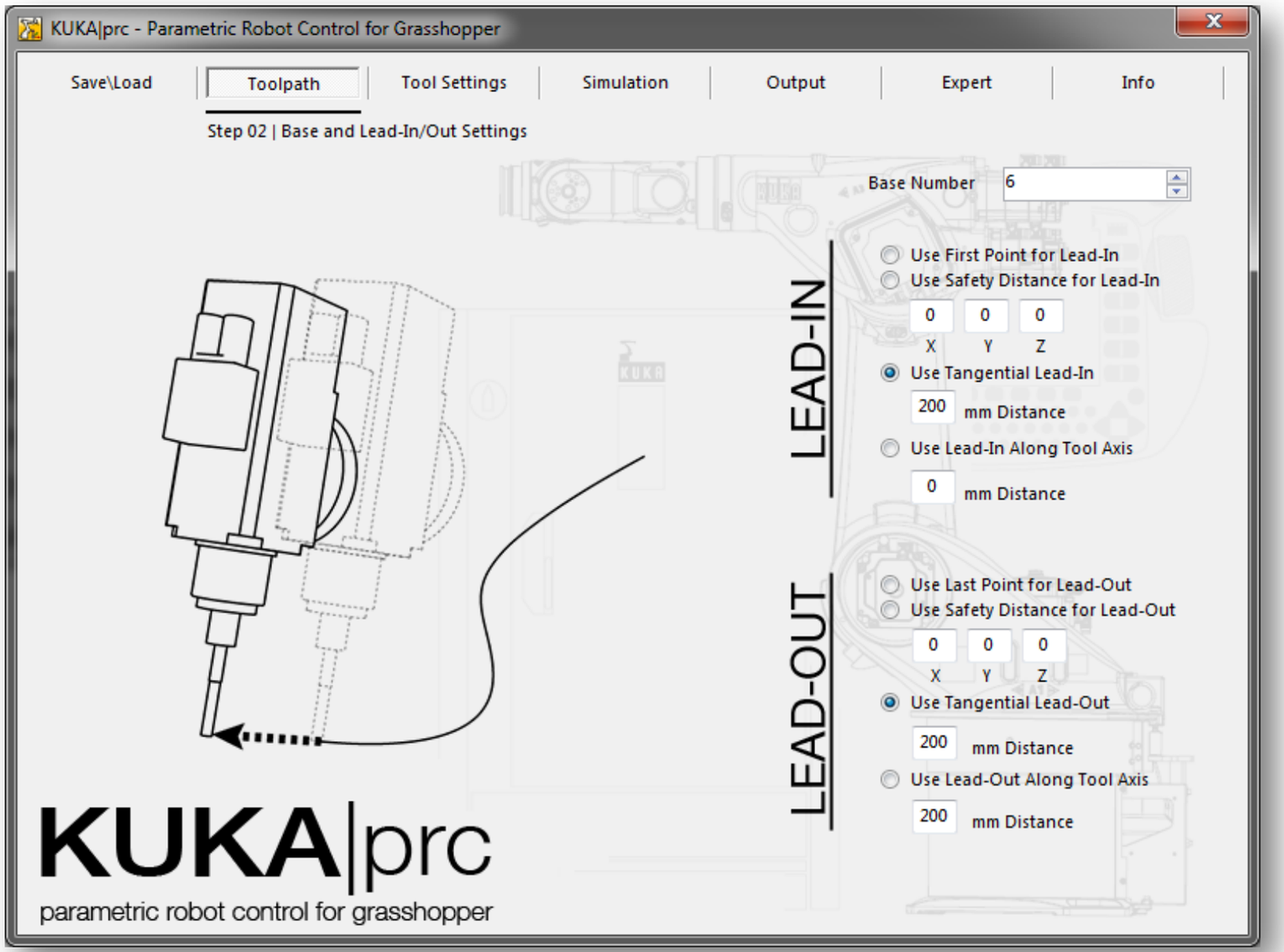




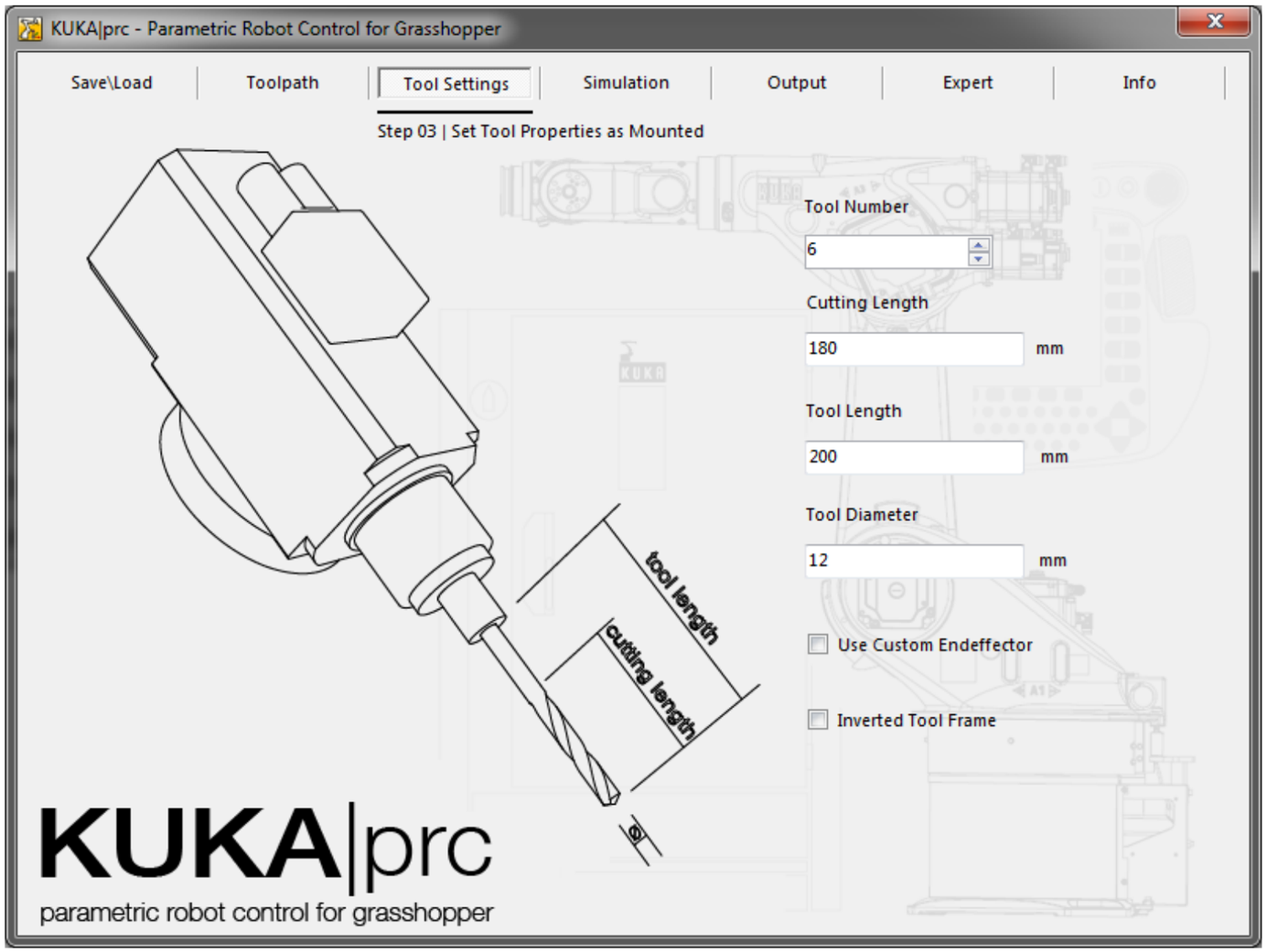
| Get the core KUKA|prc component from the KUKA|prc tab and connect the outputs with the corresponding inputs. Then right-click the centre of the component and choose KUKA|prc settings.



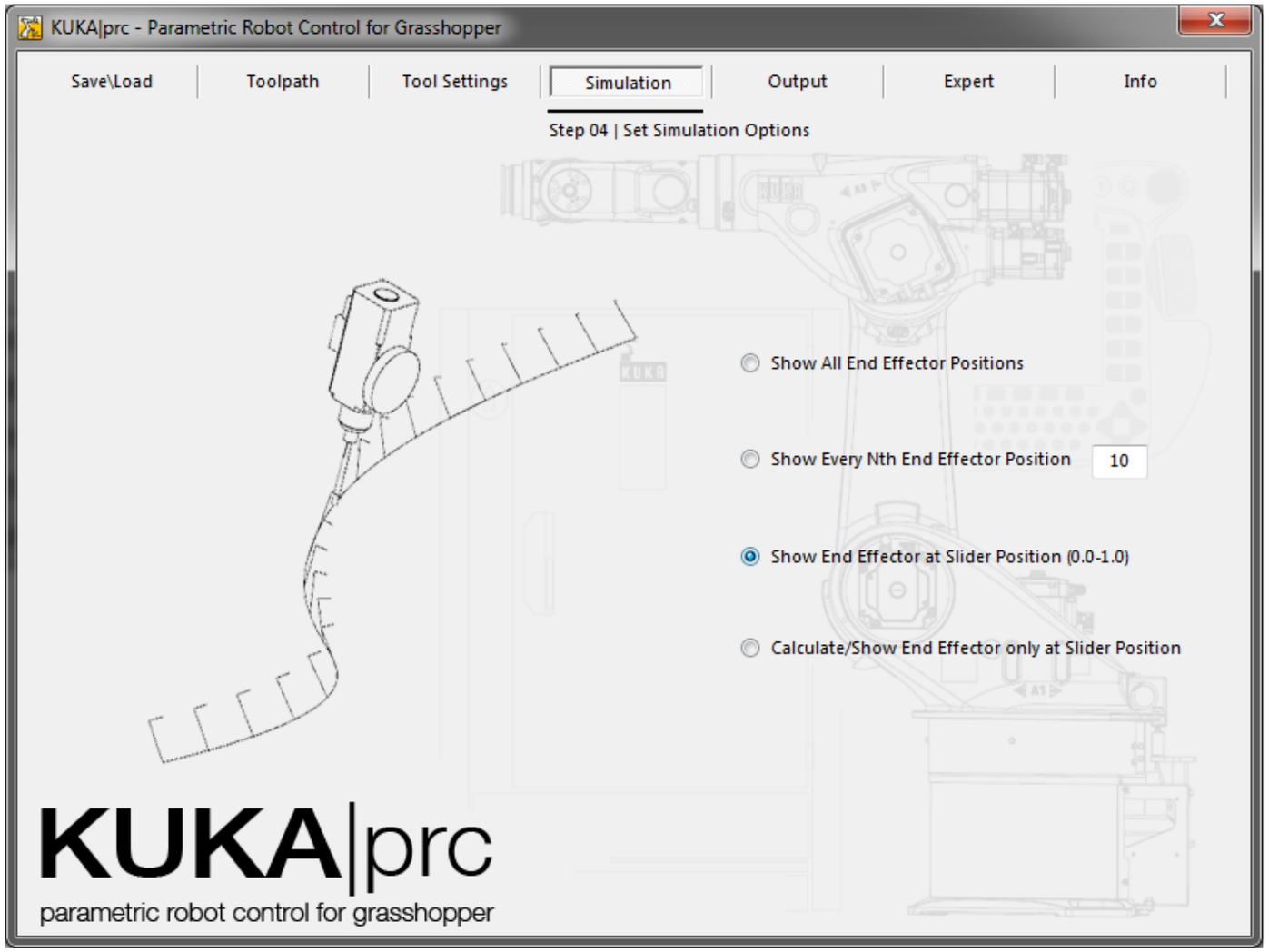
| You are now in the KUKA|prc interface. First go to toolpath and set the base number to 6. Below, you can choose the approach and retraction strategy. For this project, choose a tangential lead-in and lead-out with 200mm distance.



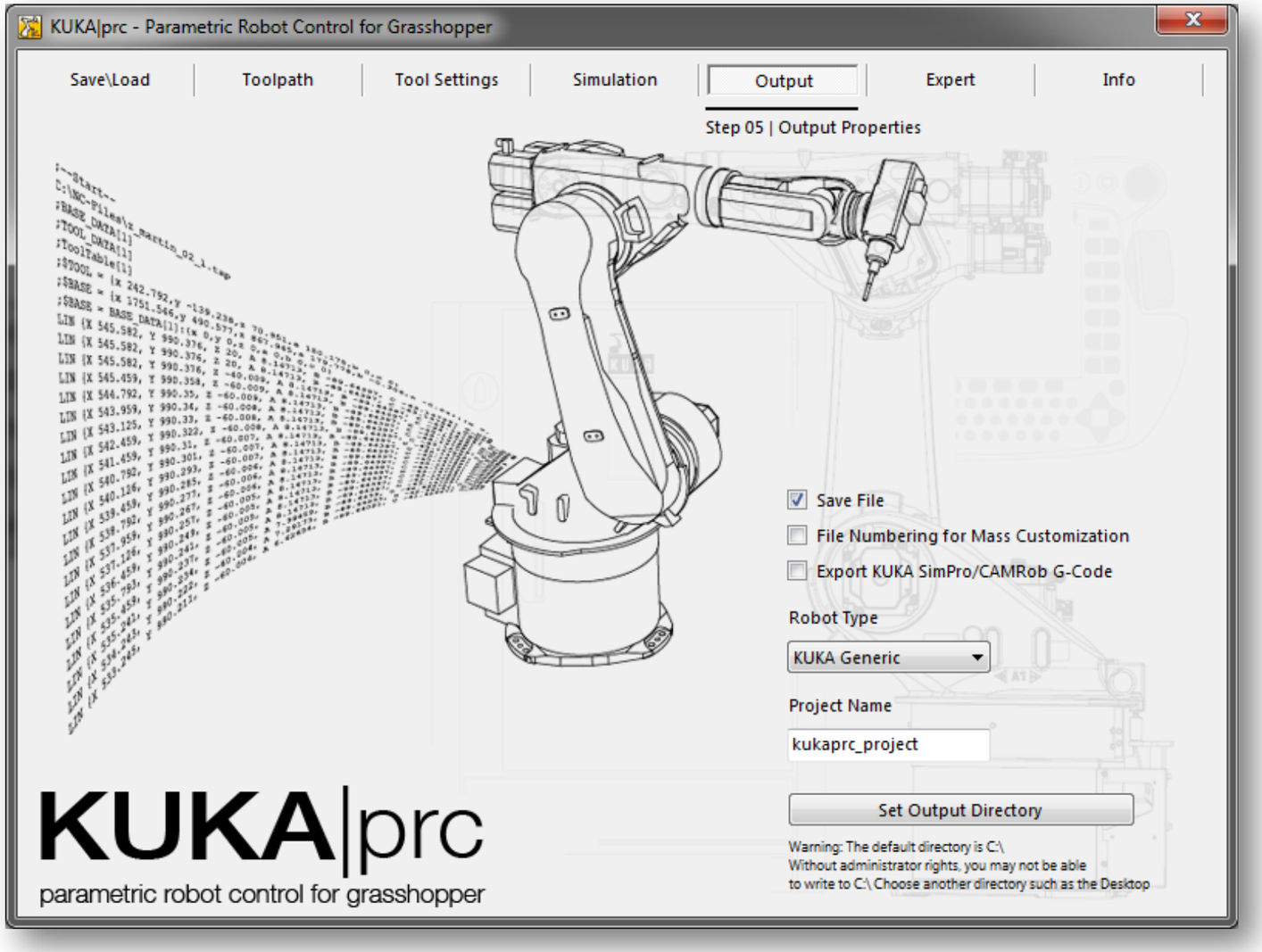
| In the next tab, set the properties of the tool. For this job, a tool with 200mm length, 180mm cutting length, and 12mm diameter seems like a good choice. Set the tool number to 6.



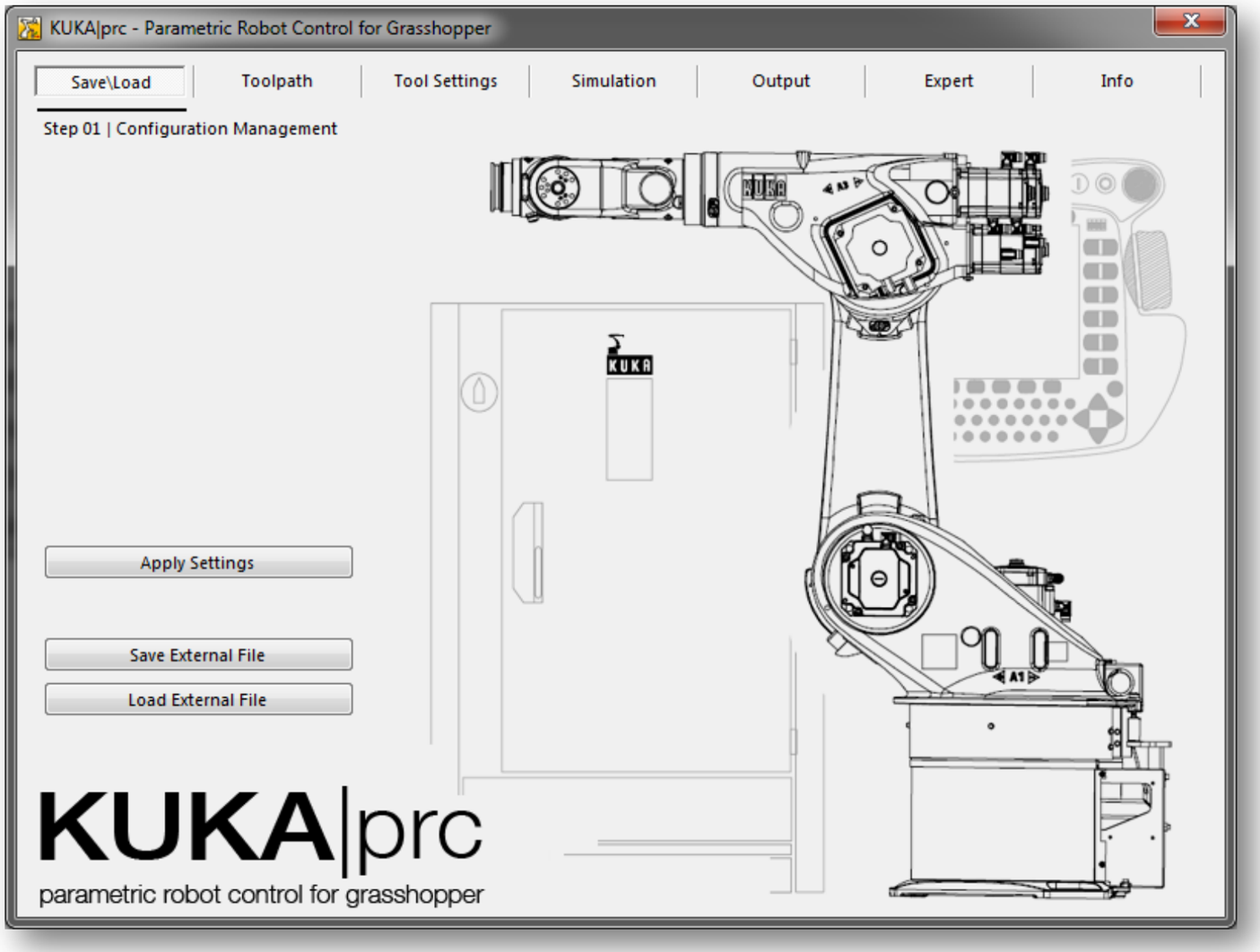
| Choose your simulation type in the next tab. Warning: Show all end-effectors might crash your PC, if there are too many points. For now, go with the third option.



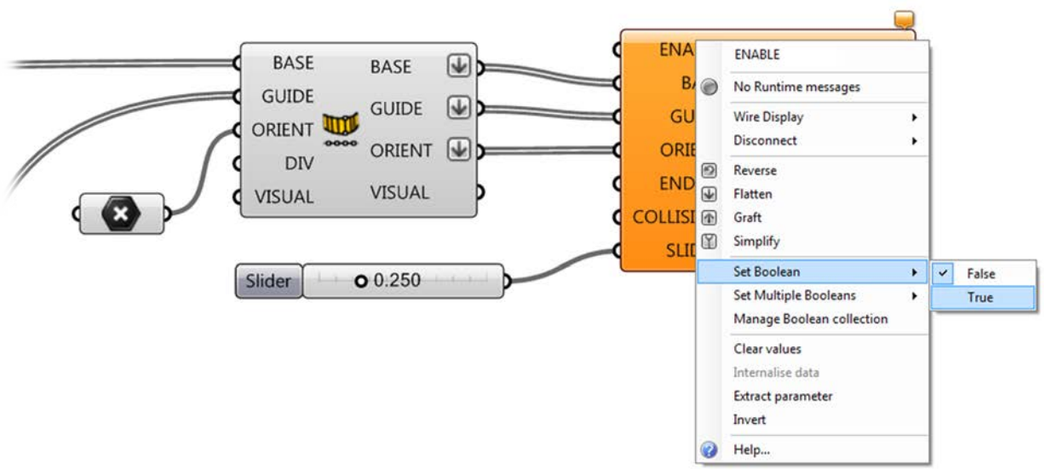
| In the output tab, give the project a proper name and set the output directory e.g. to your desktop.

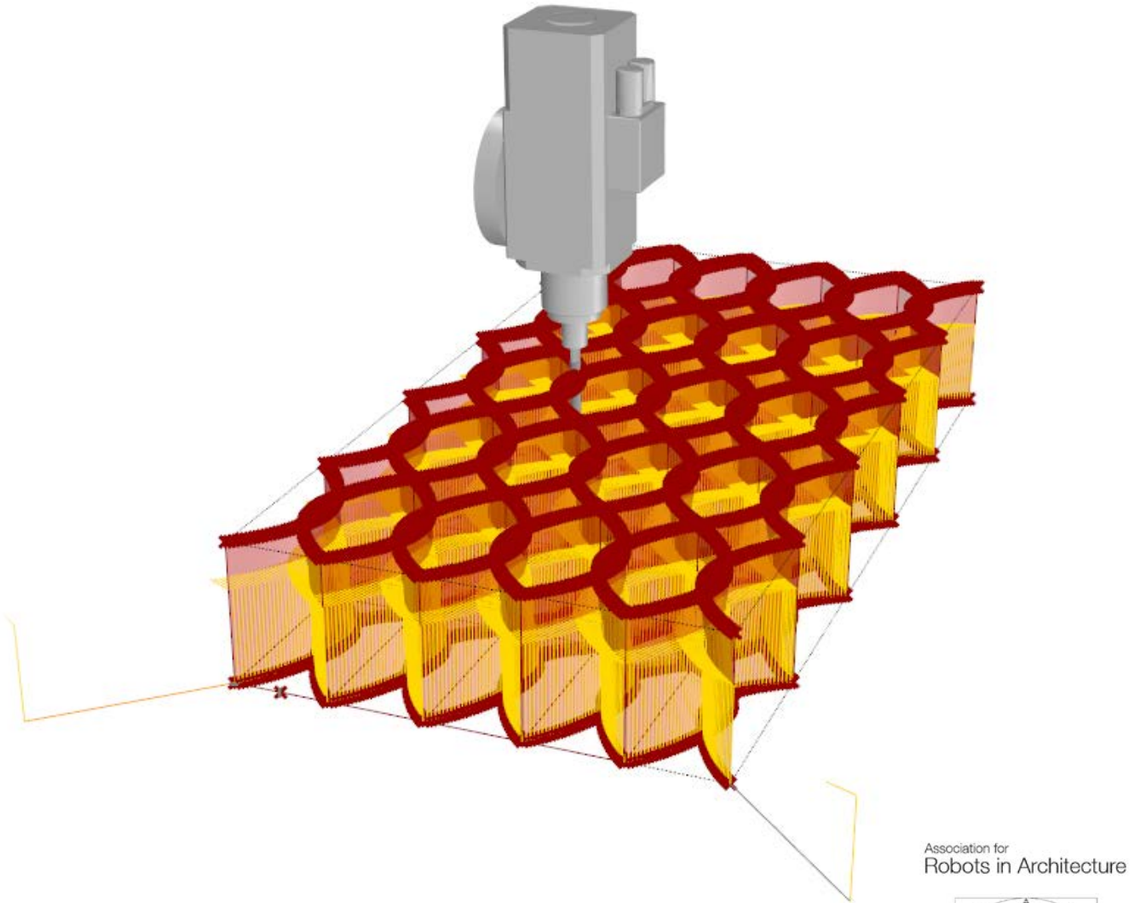


| Now go back to the Save/Load tab and click Apply Settings. You can now close the window and go back to Grasshopper.

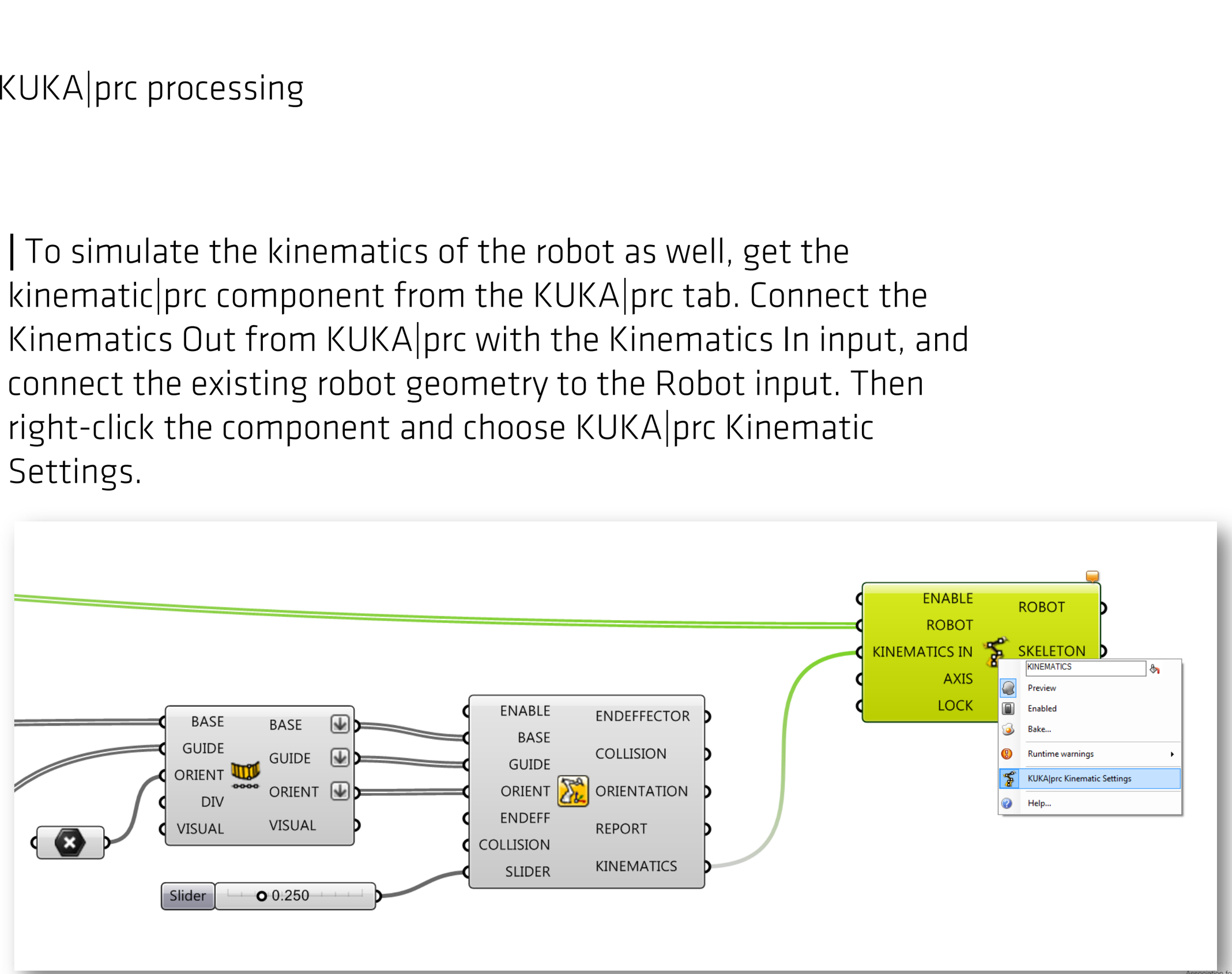


| Back in Grasshopper, attach a standard numeric slider to the the Slider input of KUKA|prc. Then right-click on Enable and set the Boolean to true. By pushing the slider, you can now simulate the robot tool when it mills the stockmodel.

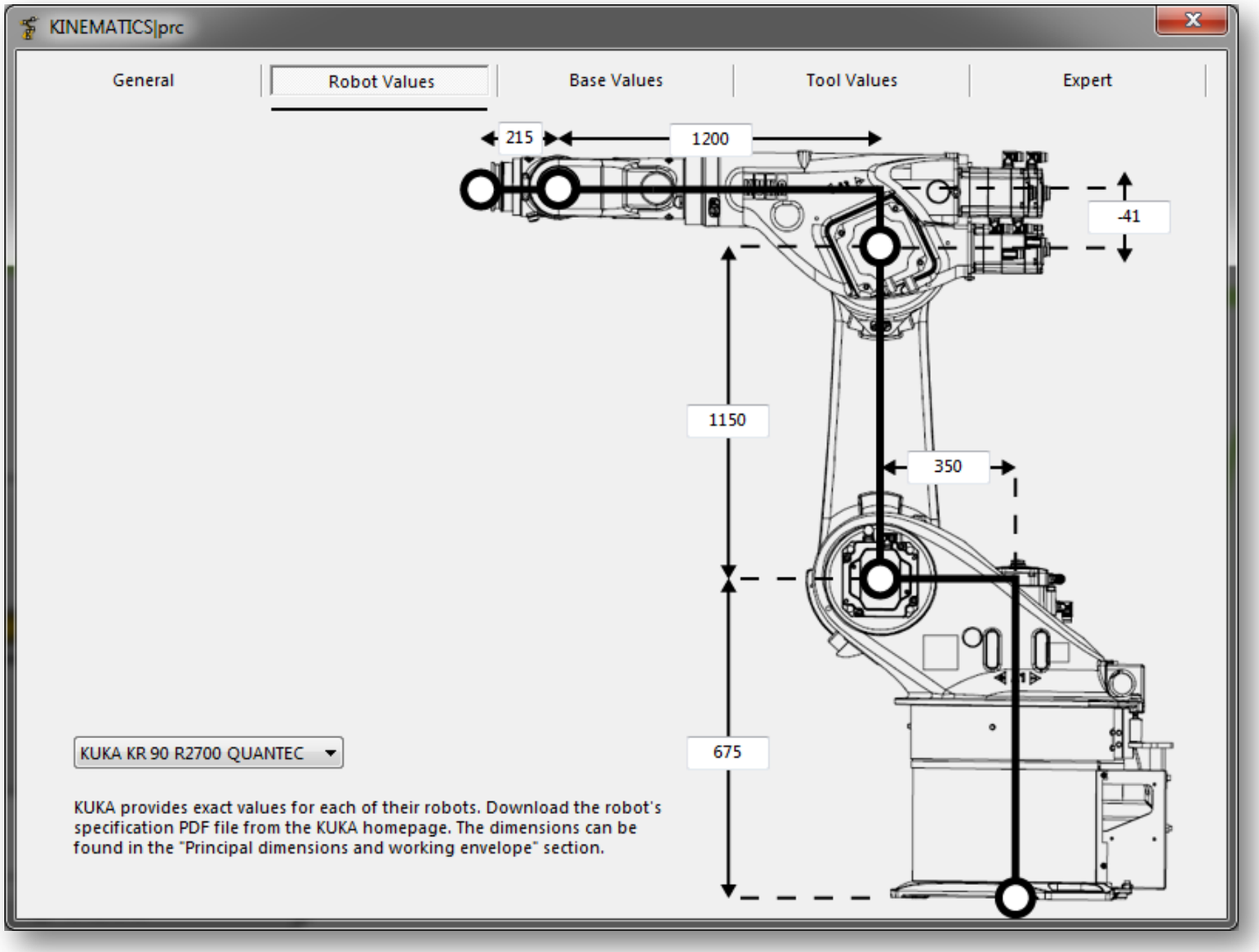




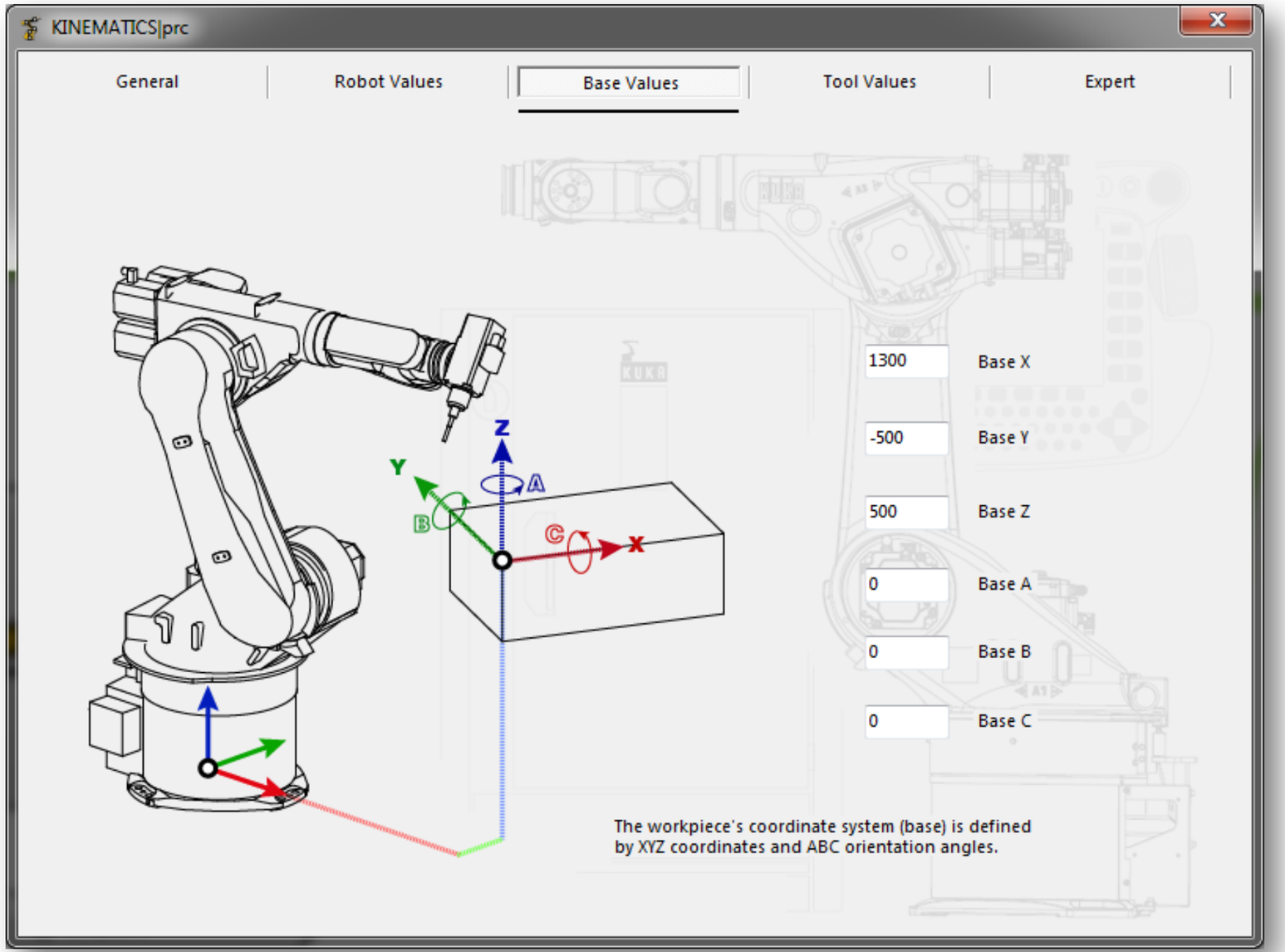
| To simulate the kinematics of the robot as well, get the kinematic|prc component from the KUKA|prc tab. Connect the Kinematics Out from KUKA|prc with the Kinematics In input, and connect the existing robot geometry to the Robot input. Then right-click the component and choose KUKA|prc Kinematic Settings.



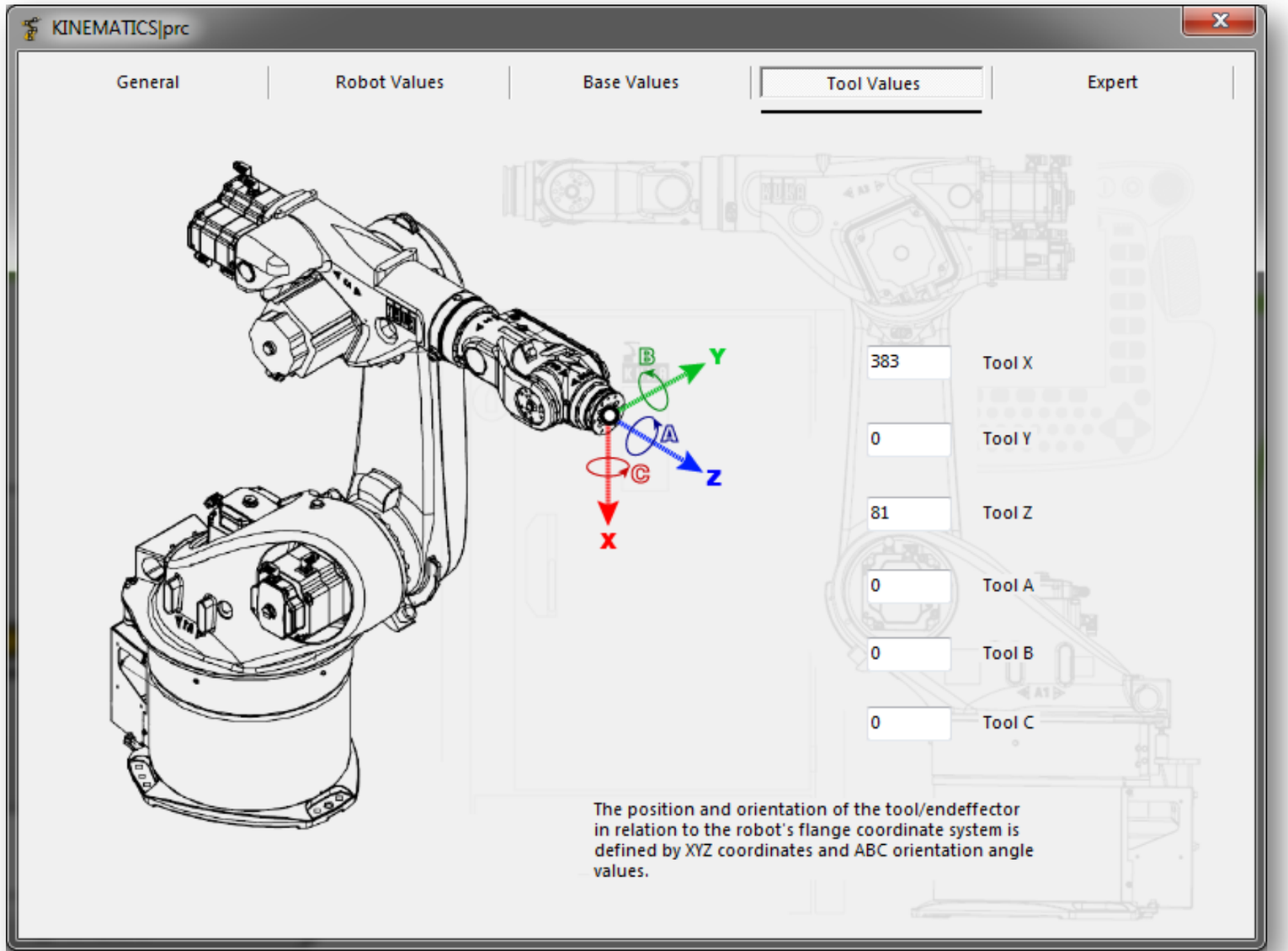
| In the Kinematics interface, first go to Robot Values and choose KUKA KR90 R2700 Quantec in the preset dropdown menu.



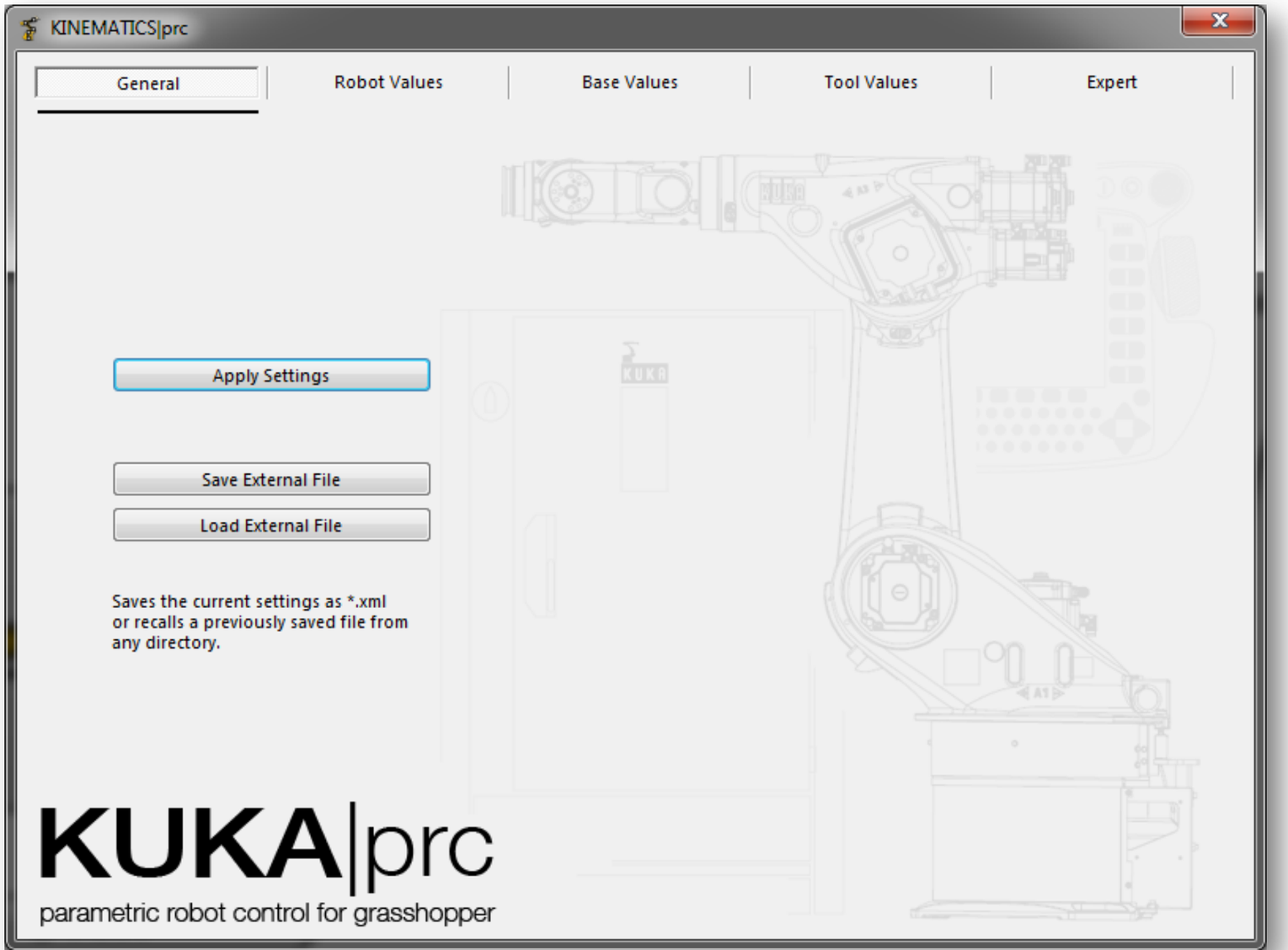
| Next up are the base settings, which define where Rhinos coordinate system is in relation to the robot's global coordinate system. Set X to 1300, Y to -500 and Z to 500.



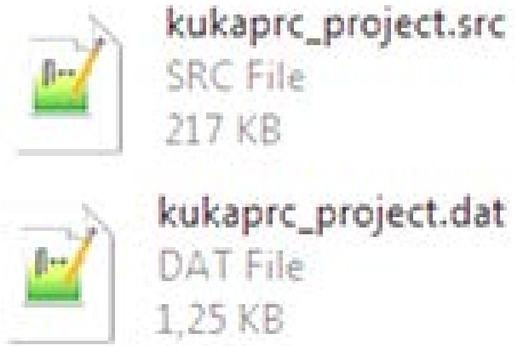
| Finally, you have to define where the tooltip is in relation to the robot's flange. Our spindle with the 200mm tool has the following values: X = 383, Y= 0, Z=81.

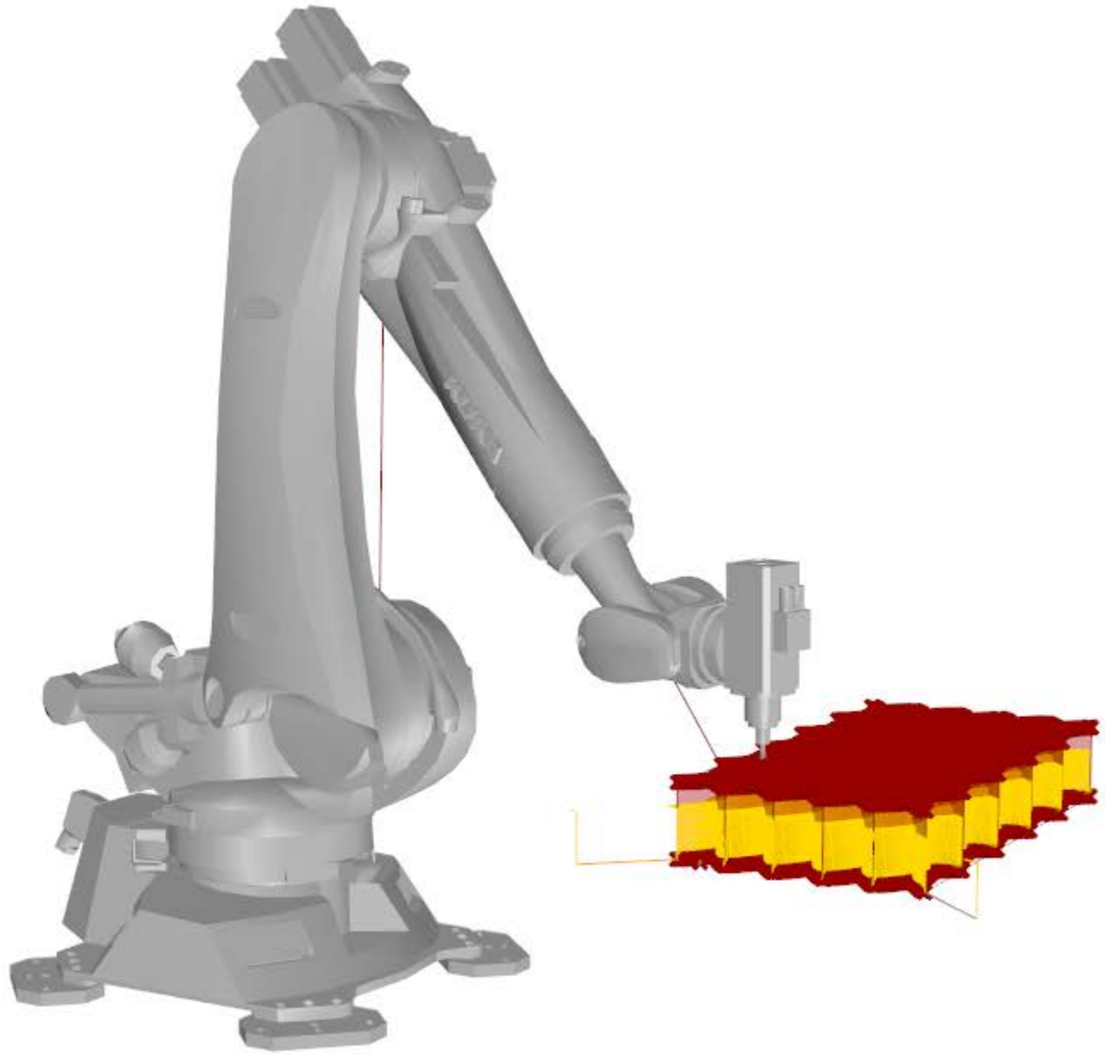


| Again, apply the settings and close the window. Set Enable again to True. Move the slider to simulate the complete robot movements.

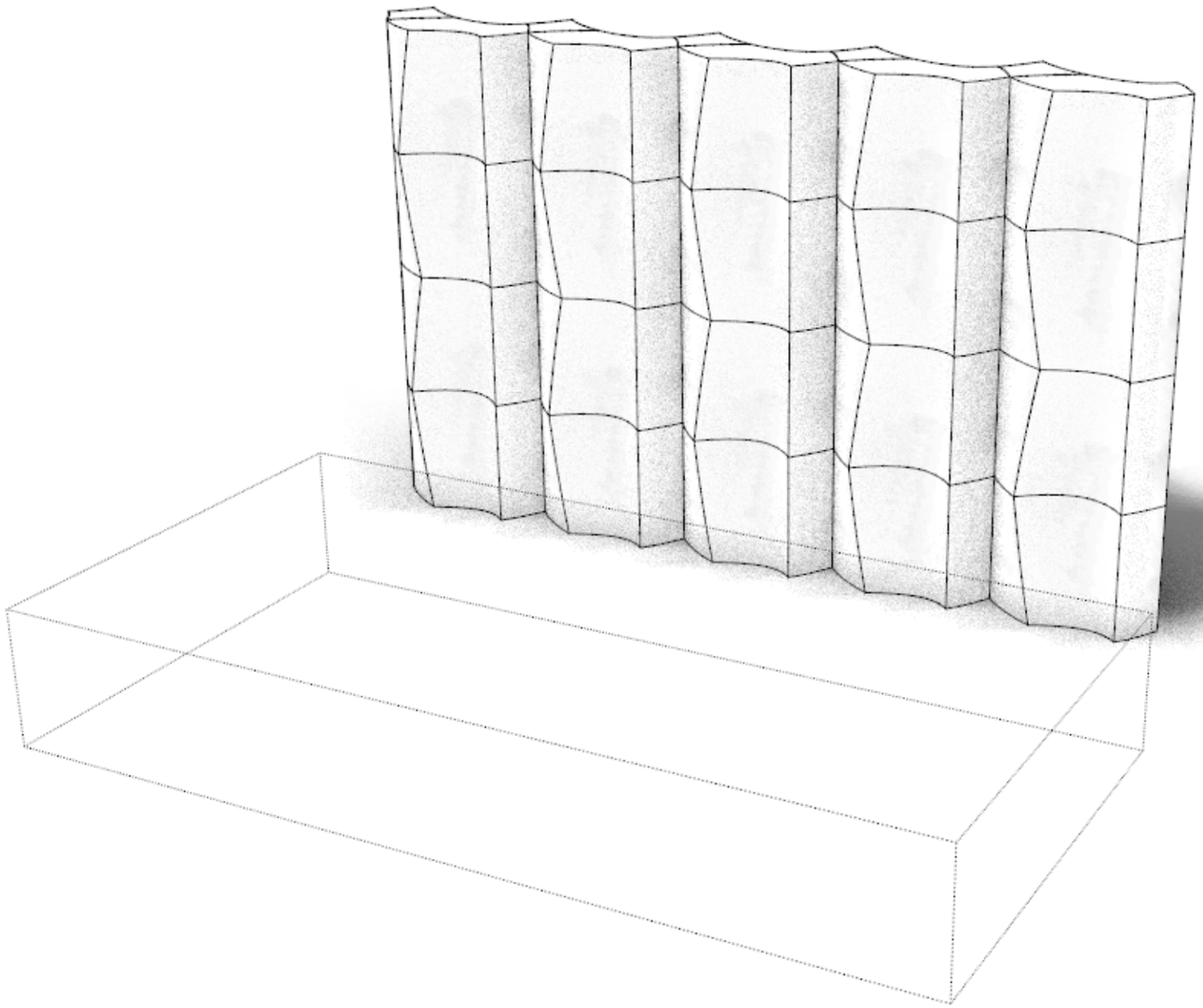


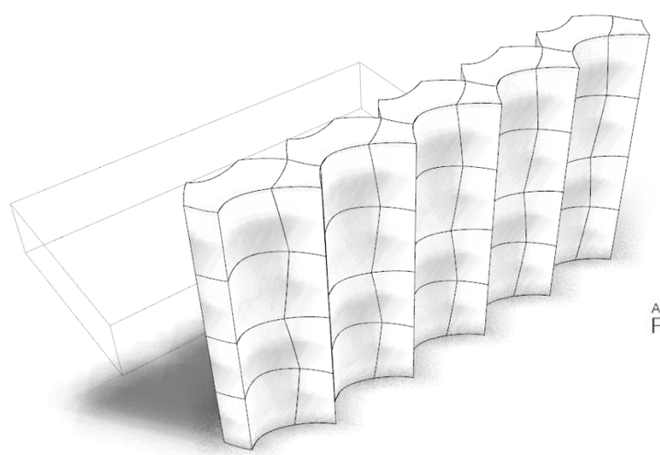
| Whenever you change something in the code, a new file gets written to your harddrive. The robot control data consists of two separate files, a *.src file and a *.dat file.



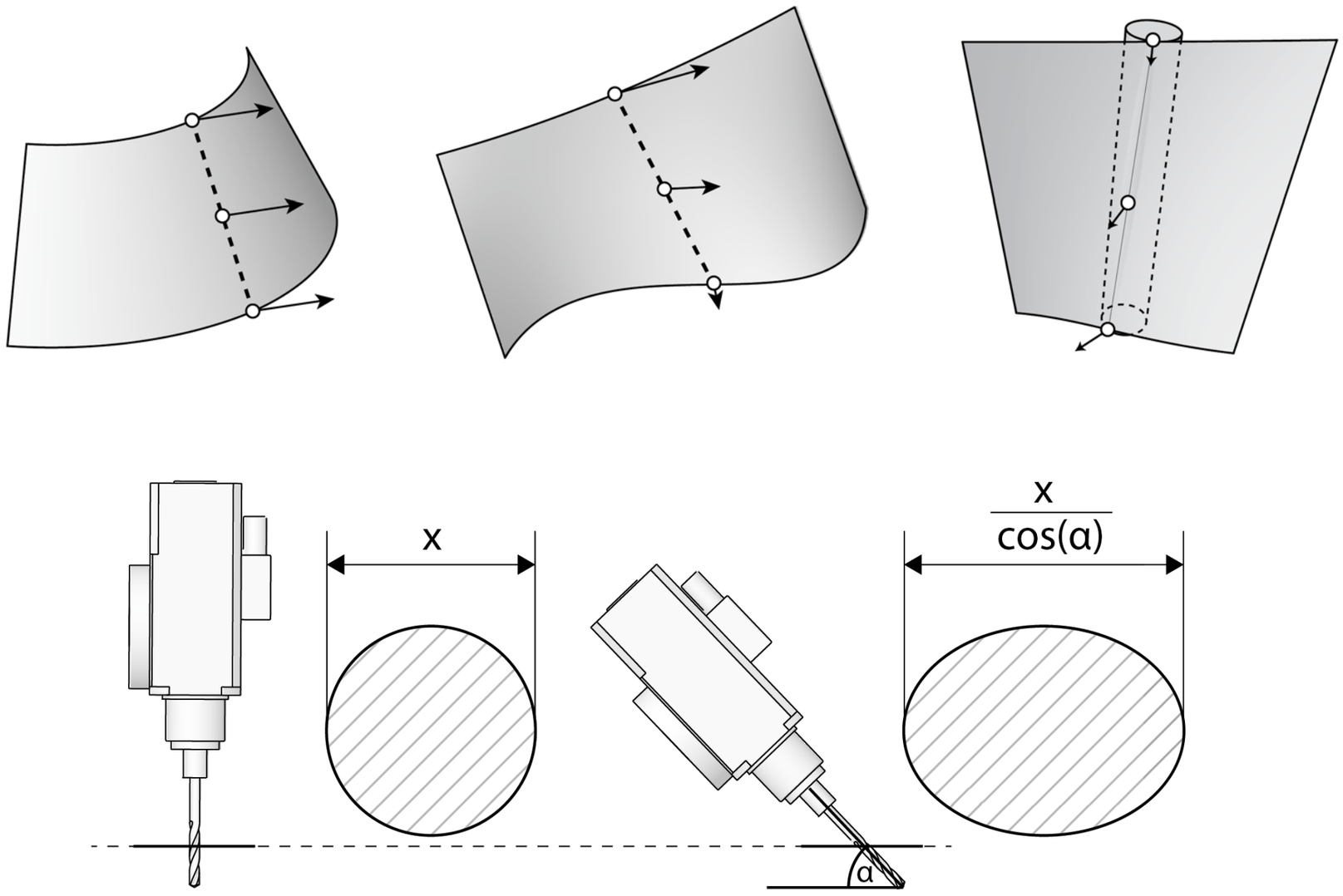


| Ideally, such a project would result in these pieces, which can be adaptively stacked on top of each other.





| Why won't there be such an ideal result? Why did the former image require two blocks instead of one?



Introduction to **GRASSHOPPER** using

KUKA|prc

parametric robot control for grasshopper

12/2012: Rob|Arch 2012 Conference

Workshop registration opening soon

Association for Robots in Architecture

ROB|ARCH 2012

Robotic Fabrication in Architecture, Art, and Design

14. 12. – 16. 12. 2012
Conference Robot Workshops
Rotterdam by TU Delft & University of Michigan
Stuttgart by University of Stuttgart
Graz by TU Graz
Zurich by ETH Zurich & ROB Technologies
Vienna by TU Vienna
Vienna by HAL

17. 12. – 18. 12. 2012
Conference Sessions
TU Vienna/Austria

www.robarch2012.org
chair@robarch2012.org