Ornamental Turning Workbench

Version 1.0

Alan Battersby

Table of Contents

Introduction

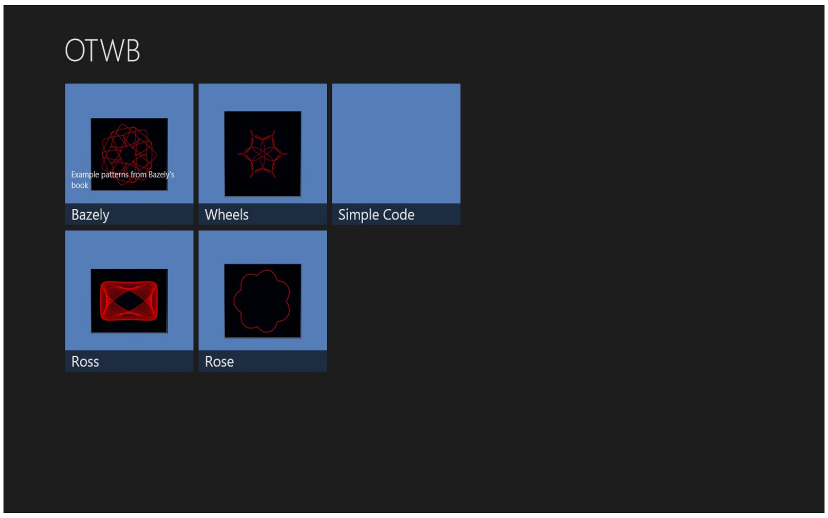
Ornamental turning is a process of cutting complex patterns on materials such as wood, metals and plastics.

One method of doing this is called **Rose Engine Turning** which uses a special lathe upon which a rotating pattern called a rosette is used to rock the lathe headstock from side to side as it rotates. By this means the tool can be made to cut a non circular pattern on the work piece held on the headstock spindle even though this work piece is itself moving in a circular path around the spindle centre. The rocking motion will move the position of the centre of the headstock relative to the tool so that the tool cuts at different radial values as the work rotates.

Another method is to attach a complex device called a **Geometrical chuck** to the spindle of a standard lathe. The geometric chuck is made so that its centre of rotation moves as the spindle rotates. This again causes a non circular path to be cut by a tool in a fixed position.

When simulating this motion on a CNC based system we want to generate the path the cutter would trace on the work piece as a sequence of points. From this sequence of points we can then generate the G code instructions to control our CNC system.

The OTWB program has several options for generating paths displayed on the opening page. Choosing an option will send you to a page dedicated to that option.



Currently there are four options for generating paths

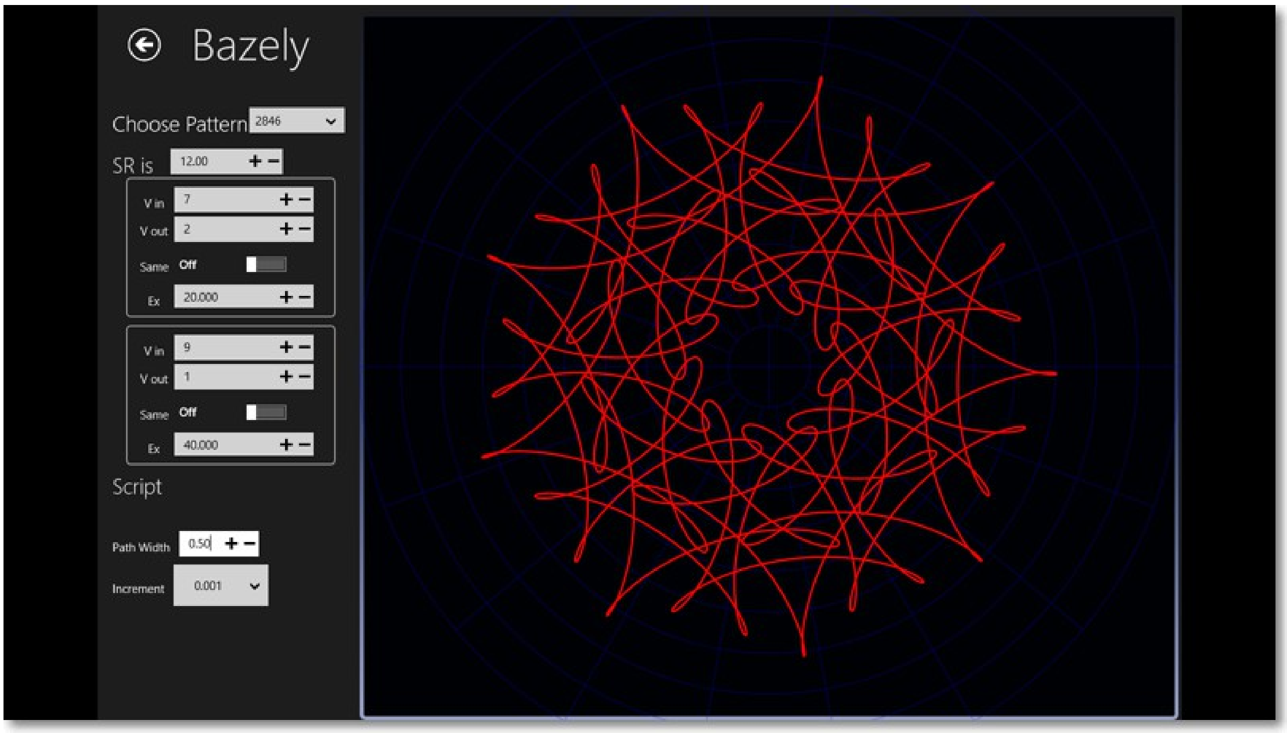
1. Geometric Chuck patterns following Thomas Bazeley
2. Patterns developed by Ross

3. Patterns developed by following the Wheels in Wheels approach
4. Patterns developed by utilising virtual rosettes on a Rose Engine

The final option takes you to the G code generation page.

Geometric Chuck

Choosing this option on the main menu allows you to choose one of the many patterns discussed in Thomas Bazely's book.

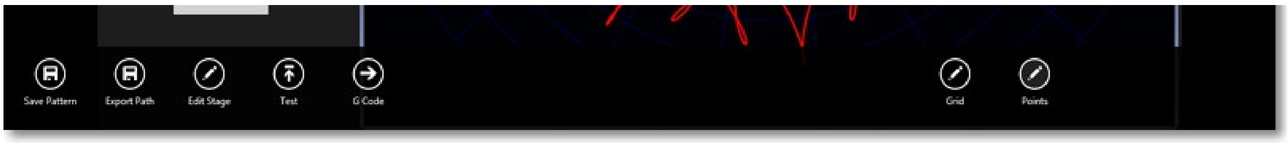


Choose Pattern: 2846
SR is: 12.00
V in: 7
V out: 2
Same: Off
Ex: 20.000
V in: 9
V out: 1
Same: Off
Ex: 40.000
Path Width: 0.50
Increment: 0.001

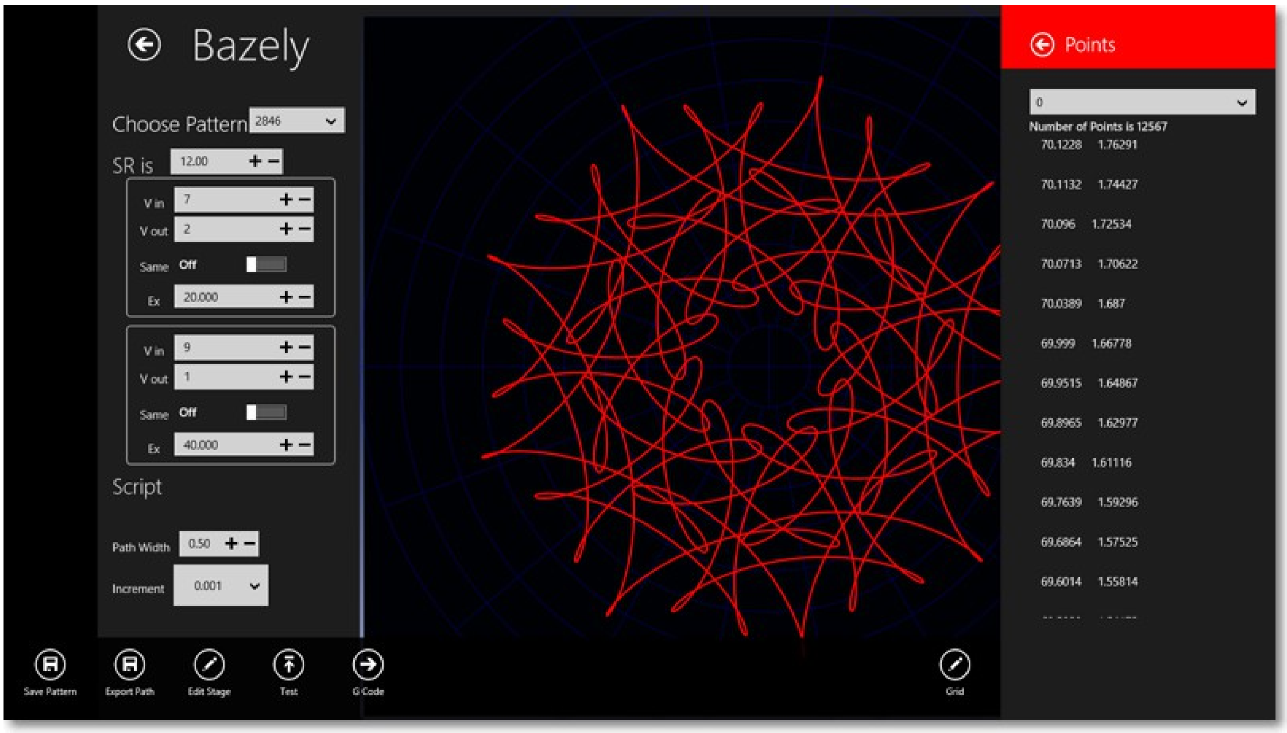
Above you see an image of the Bazely page showing pattern #2846 there are 3453 patterns described altogether. This covers most of the figures in Bazely's book that can be generated by a single run of the chuck. A run of the chuck is one where all the adjustments are constant. However several patterns in his book are the result of multiple runs with adjustments in between so these patterns are not reproduced accurately (at present) by this software.

You choose the pattern from the Choose Pattern Combo box at the top of the page and the values of the chuck settings are displayed underneath. These settings can be adjusted and the path display is updated to reflect this. You can export these new values to your disc if you want to keep them.

You can also inspect the values of the points along the path by right clicking to see the bottom App bar and then clicking on the points button.



This will cause the points inspector flyout to appear


Bazely

Choose Pattern 2846

SR is 12.00

V in 7
V out 2
Same Off
Ex 20.000

V in 9
V out 1
Same Off
Ex 40.000

Script

Path Width 0.50

Increment 0.001

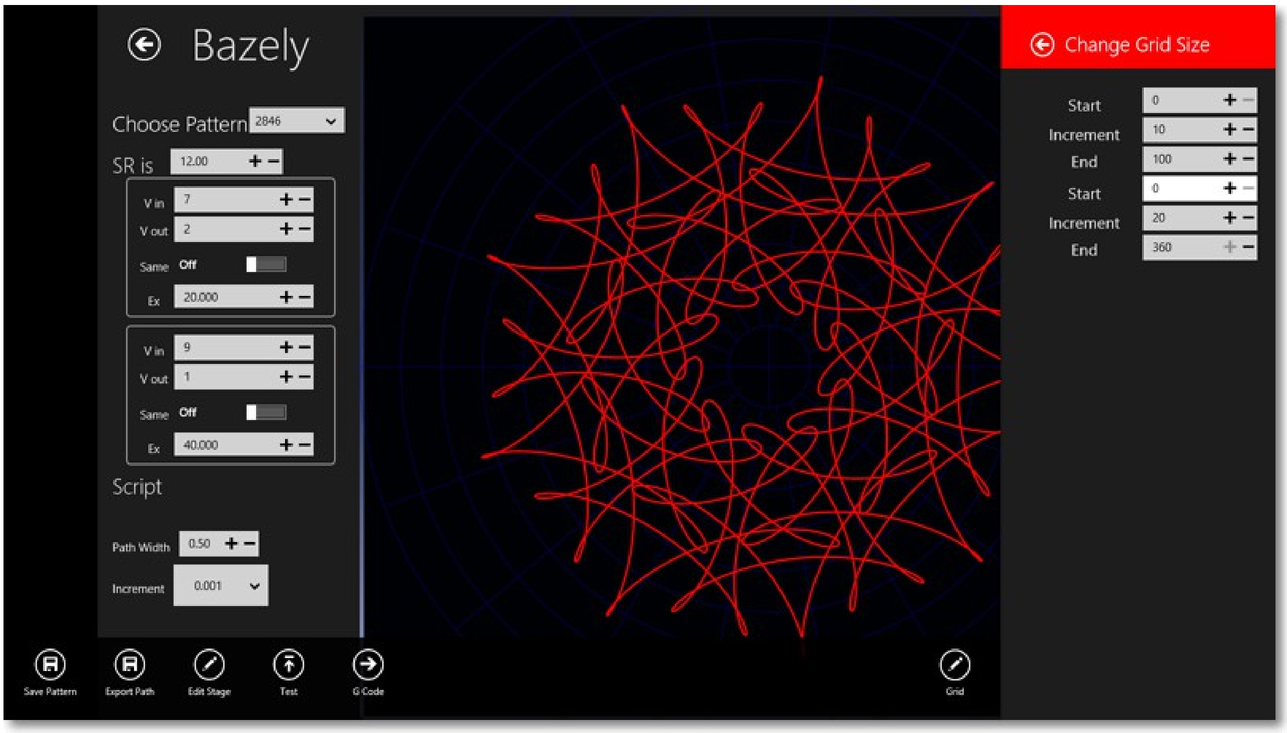
Points

0

Number of Points is 12567

```
70.1228   1.76291
70.1132   1.74427
70.096    1.72534
70.0713   1.70622
70.0389   1.687
69.999    1.66778
69.9515   1.64867
69.8965   1.62977
69.834    1.61116
69.7639   1.59296
69.6864   1.57525
69.6014   1.55814
```

The scale of the drawing is defined in relation to the grid shown in blue. The scale is set so that the grid just fills the drawing area. You can of course adjust the grid by clicking the edit grid button located at the right hand side of the bottom App bar.


Bazely

Choose Pattern 2846

SR is 12.00

V in 7
V out 2
Same Off
Ex 20.000

V in 9
V out 1
Same Off
Ex 40.000

Script

Path Width 0.50

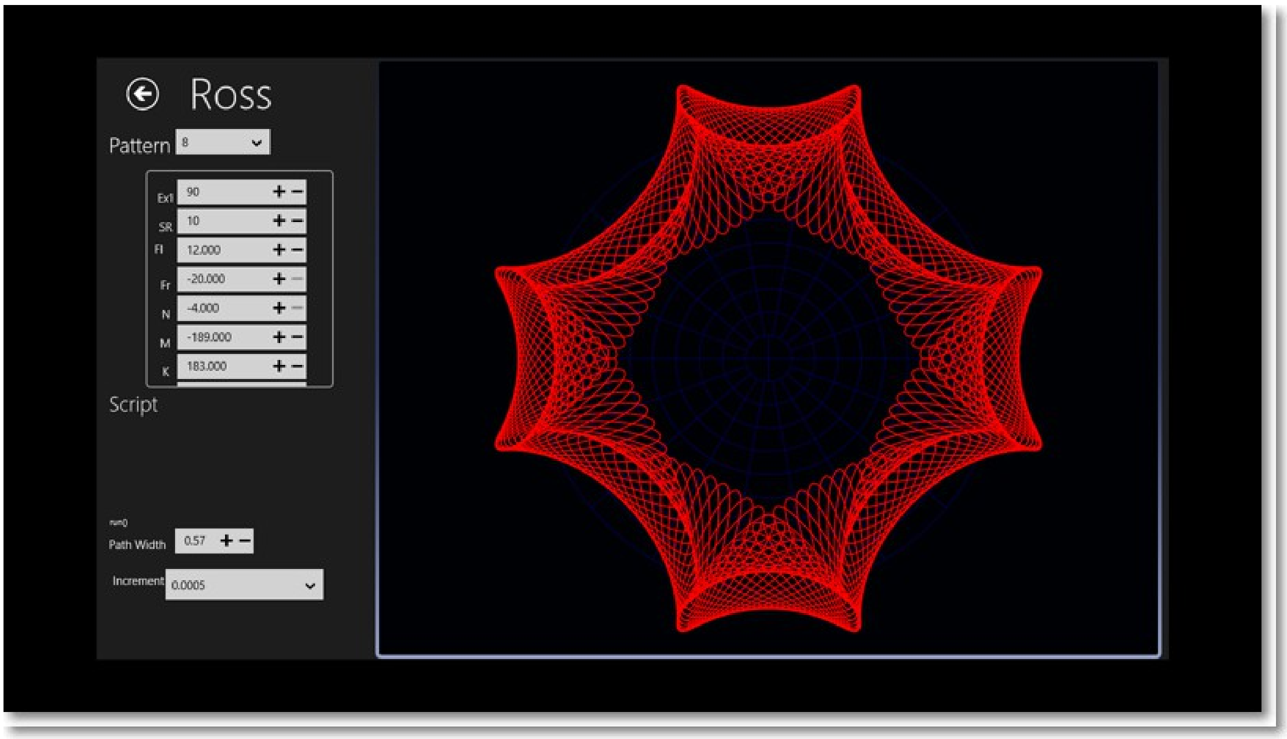
Increment 0.001

Change Grid Size

Start	0
Increment	10
End	100
Start	0
Increment	20
End	360

The top range is the Radial range for the grid, whilst the second range is the angular range in both cases the increment gives the distance between successive markings on the grid.

Ross Paths


Ross

Pattern 8

Ex1 90

SR 10

Fl 12.000

Fr -20.000

N -4.000

M -189.000

K 183.000

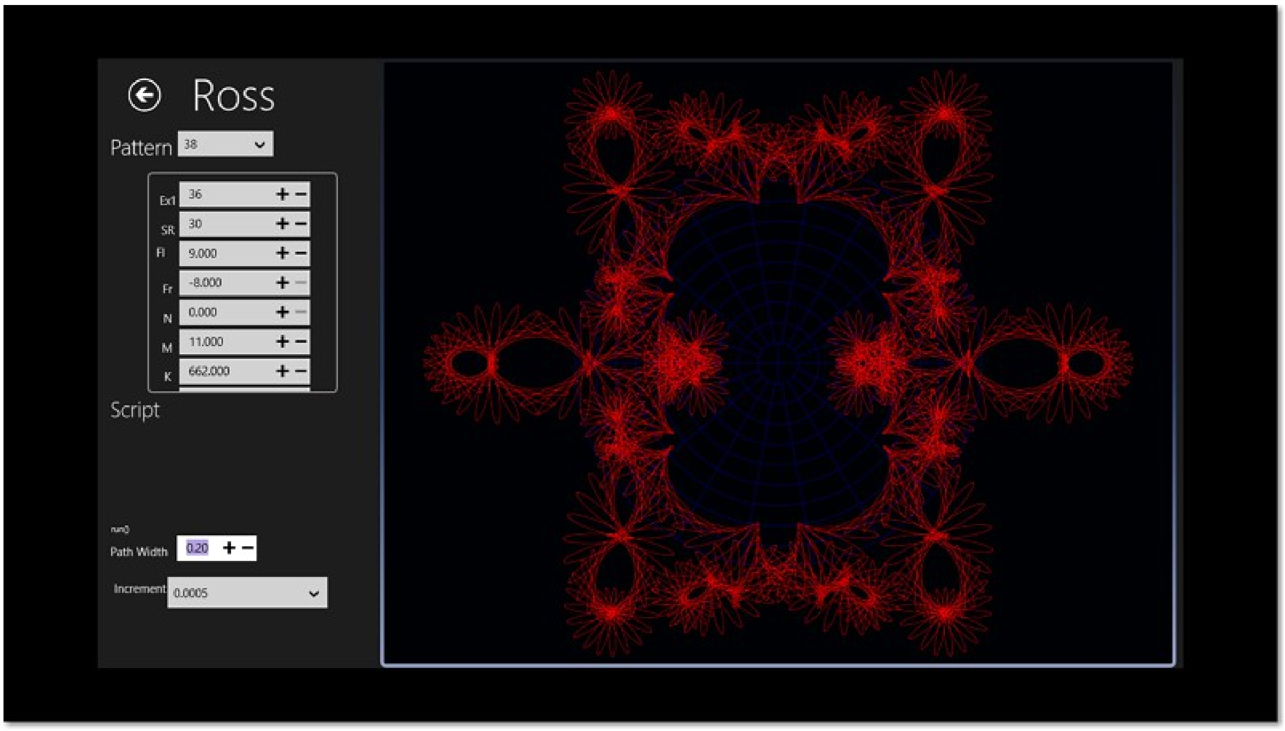
Script

run()

Path Width 0.57

Increment 0.0005

The page describing the Ross paths is similar to the Bazely page and has the same functionality. You can adjust the initial values and export them to file, then subsequently import them during another session. There are 38 Ross Patterns the image below is the last pre-defined pattern


Ross

Pattern 38

Ex1 36

SR 30

Fl 9.000

Fr -8.000

N 0.000

M 11.000

K 662.000

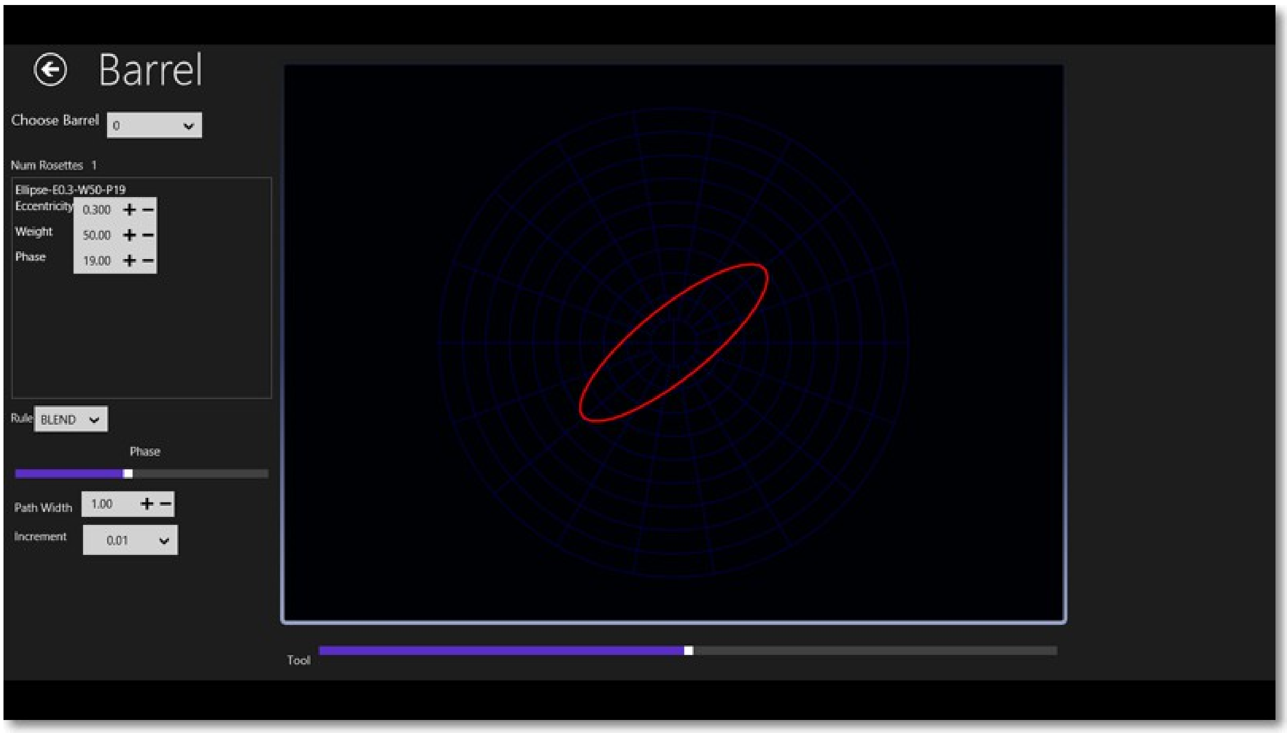
Script

run()

Path Width 0.20

Increment 0.0005

Rose Engine


Barrel

Choose Barrel 0

Num Rosettes 1

Ellipse-E0.3-W50-P19
Eccentricity 0.300
Weight 50.00
Phase 19.00

Rule BLEND

Phase

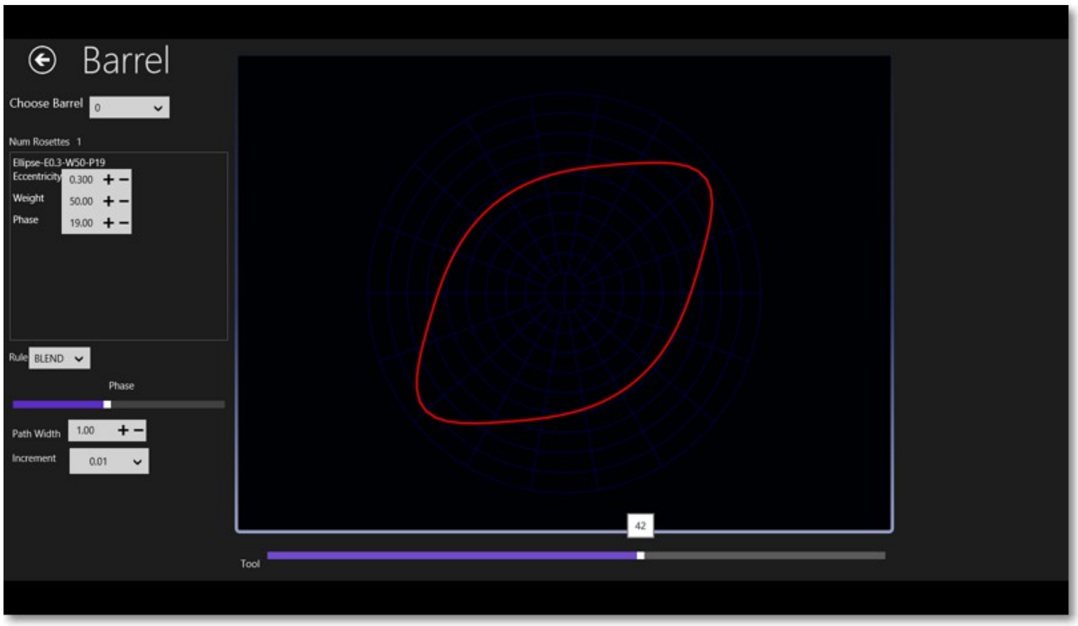
Path Width 1.00

Increment 0.01

This page is slightly different from the others. Because the rosettes on a rose engine barrel act to offset the headstock from its central position. The values generated by this page are also offsets and not absolute positions.

Therefore the path generated by a tool is dependent on the tools position as well as the arrangement of rosettes on the barrel and finally the overall phase of the barrel.

At the bottom of the display there is a slider to allow you to change the position of the tool relative to the headstock set central position. At the side of the display is another slider to allow you to set the overall phasing of the barrel.


Barrel

Choose Barrel 0

Num Rosettes 1

Ellipse-E0.3-W50-P19
Eccentricity 0.300
Weight 50.00
Phase 19.00

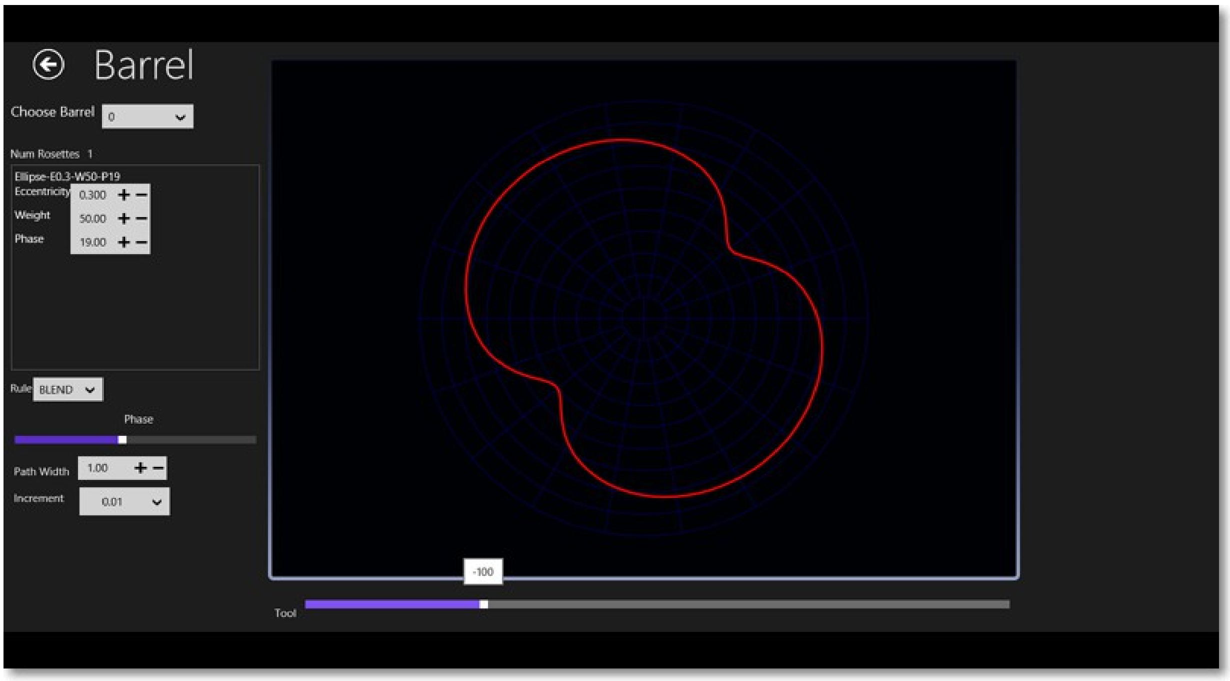
Rule BLEND

Phase

Path Width 1.00

Increment 0.01

The image above is of the tool path when the tool is offset by a value of 42mm whilst the image below shows the path created by the barrel with a tool offset value of -100mm.



Choose Barrel 0

Num Rosettes 1

Ellipse-E0.3-W50-P19

Eccentricity 0.300

Weight 50.00

Phase 19.00

Rule BLEND

Phase

Path Width 1.00

Increment 0.01

-100

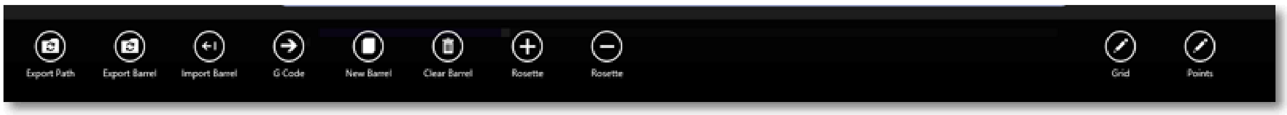
Tool

The Barrel

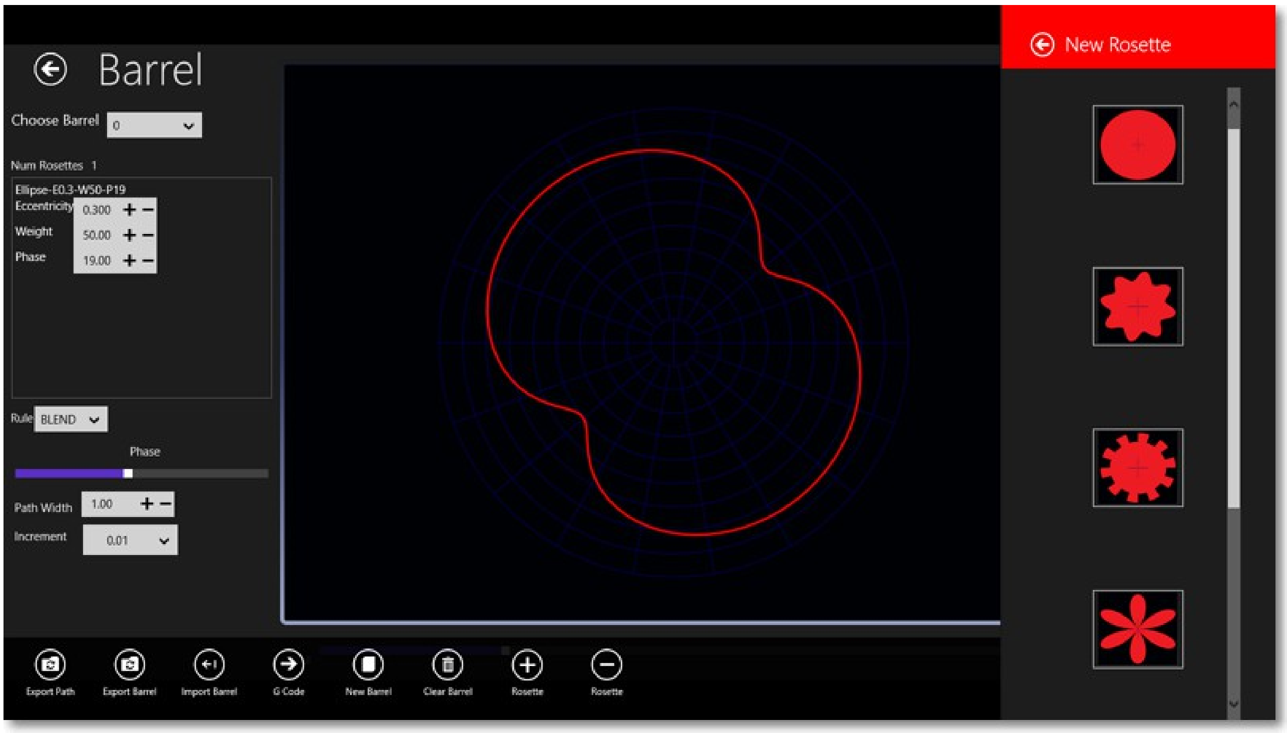
Comprises of one or more rosettes. These rosettes are displayed in the scrollable view to the side of this display. Again like the other pages the rosette parameters can be adjusted and the display is updated to reflect the changed path.

There are buttons on the bottom app bar that allow you to

1. Export the path
2. Export the barrel data
3. Import a barrel
4. Create a new empty barrel which is then automatically selected.
5. Clear the currently selected barrel this means remove all its rosettes.
6. Add a rosette (+)
7. Remove a rosette (-)
8. Change the grid size
9. View the points on the path.



When you choose to add another rosette you the choice options are displayed on a flyout to the right of the screen


Barrel

Choose Barrel 0

Num Rosettes 1

Ellipse-E0.3-W50-P19

Eccentricity 0.300

Weight 50.00

Phase 19.00

Rule BLEND

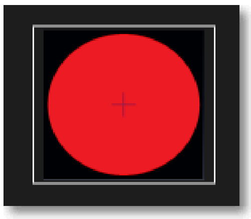
Path Width 1.00

Increment 0.01

Types of Rosette

You can choose between

An Elliptical - Circular Rosette

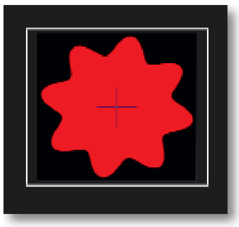


There are 3 parameters

- *Eccentricity* - this is the ratio between the two radii. A value of 1 gives a circle a value less than 1 gives an ellipse whose thickness depends on the value. So a value of 0.5 will result in an ellipse whose vertical thickness is half its width
- *Weight* - this is a multiplying factor that gives the size of the ellipse. The standard ellipse is 1 unit wide so set the weight to 20 to get one 20 units wide.
- *Phase* - this determines the rotation of the ellipse around its centre.

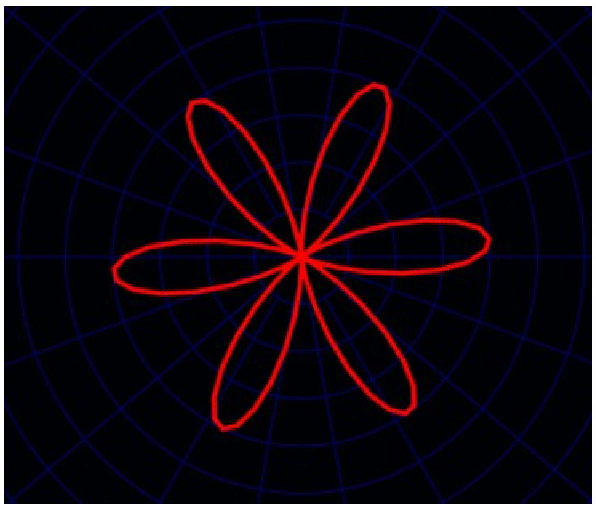
A sine wave rosette

Again three parameters

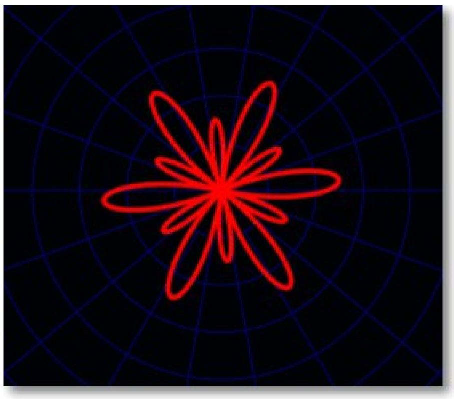


- *Frequency* – this gives the number of waves
- *Weight* – this gives the height of a wave
- *Phase* – the rotation around its centre

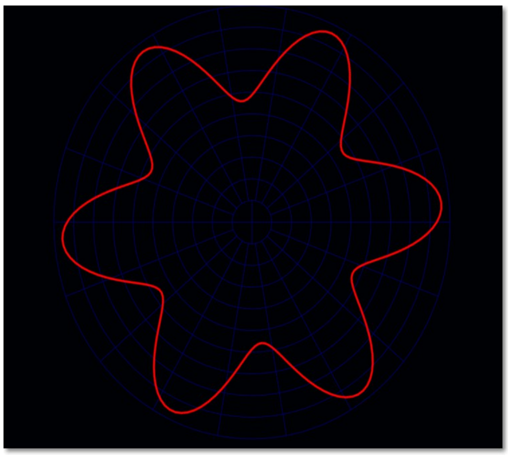
Note that if you set the tool offset equal to the weight then the path has lobes and keeps keeps going back to the origin as shown by image below.

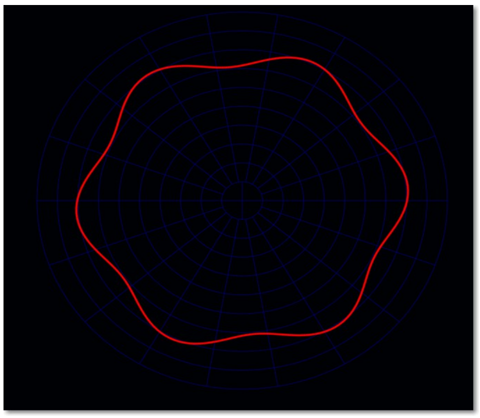


If you make the tool offset less than the weight but still positive you get alternating large and small lobes.



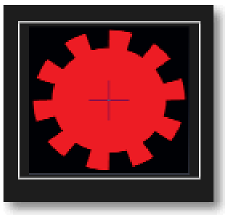
Waves with an offset value greater than the weight look as below. But note you may need to reduce the weight to quite small values to obtain a gentle ripple.



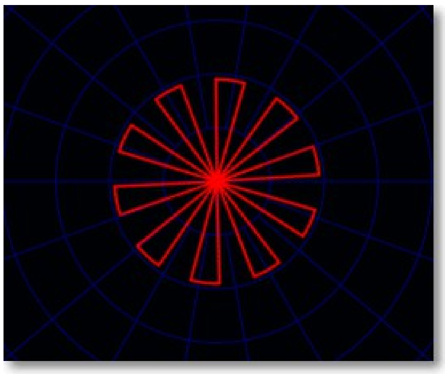


Same tool offset as the previous image with a reduced weight to give a smaller ripple.

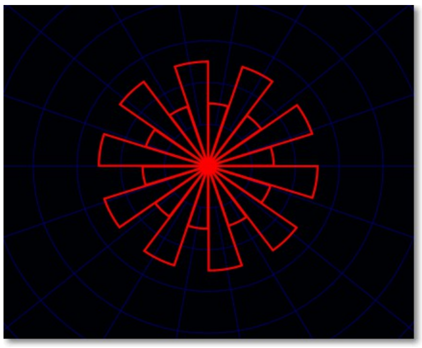
Square wave rosette



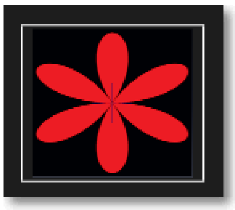
The square wave rosette has the same three parameters as the sine wave and it also behaves in the same way as it is a also a wave (though one with steep sides)



In the above image the tool position is set equal to the rosette weight so path keeps returning to the zero position. The image below shows what happens when the tool offset is less than weight of the rosette.



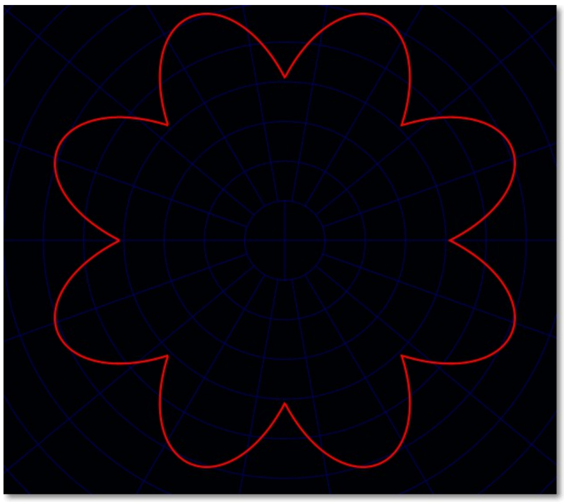
A petal rosette



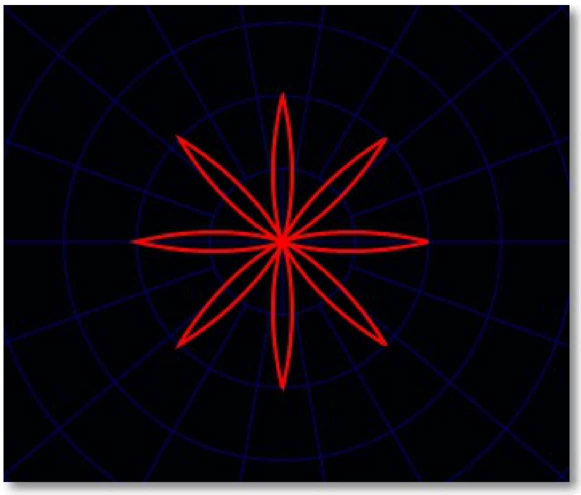
Similar to the standard wave it is in fact an absolute wave. That means when on the ordinary wave the values start to go negative on the petal they stay positive. It has the same three parameters

- *Frequency* – this time there are twice the number of petals as the frequency value
- *Weight* – same as wave
- *Phase* – same as wave

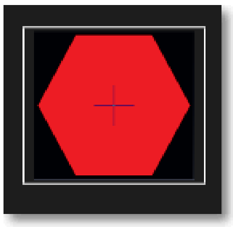
Again the tool position determines the shape of the path generated in one rotation. However unlike the wave the bumps are steeper and meet at a cusp.



By setting the tool position to negative of the weight you get a petal with sharp this sides



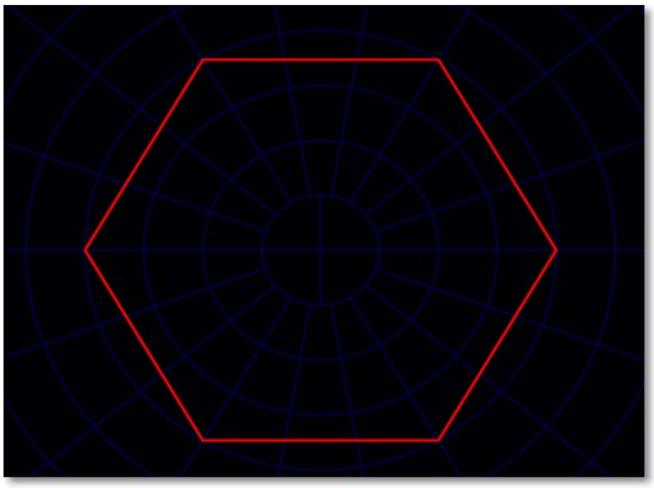
Polygon rosette



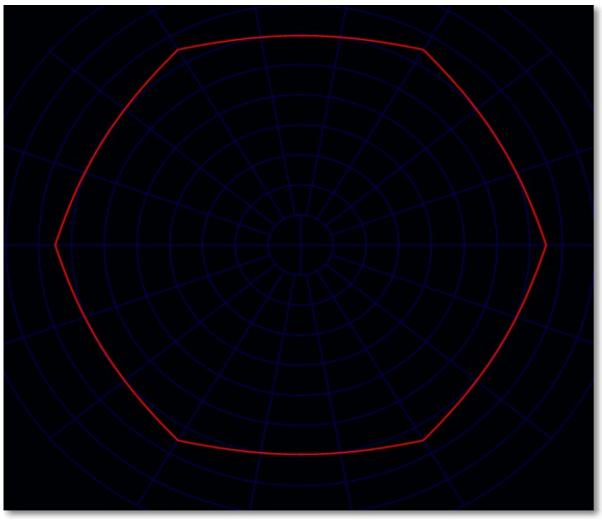
Parameters are

- *Number* of sides
- *Weight*
- *Phase*

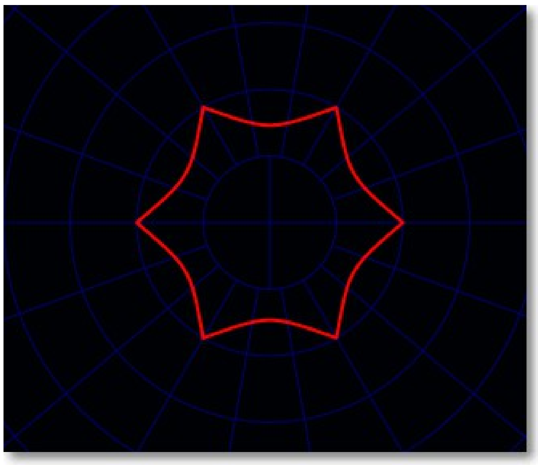
Again tool position is important. Setting the tool position to zero gives flat sides



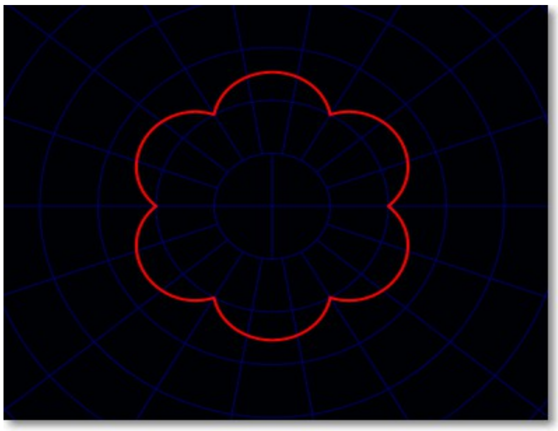
However increasing the tool position tends to make the sides more convex



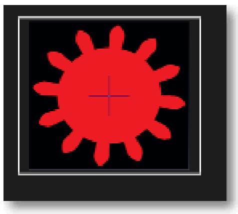
Here the weight is 40 whilst the tool position is set at 35. Alternatively decreasing the tool position makes the sides more concave so long as the tool position is between zero and – the weight.



In the image above the weight is 40 and the tool position is -20. You get a different effect when the tool position is less that the negative weight value. The image below is for a weight of 40 and a tool position of -60.



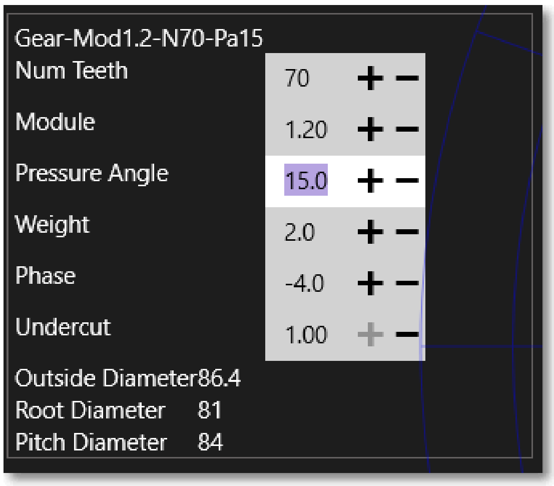
Gear Rosette



This is still very experimental and the code is not quite correct. It produces the outline of a spur gear. This rosette is totally different from the others because its size is not dependant on tool position. Instead the size of the gear path depends upon a standard set of parameters shown below.

parameters

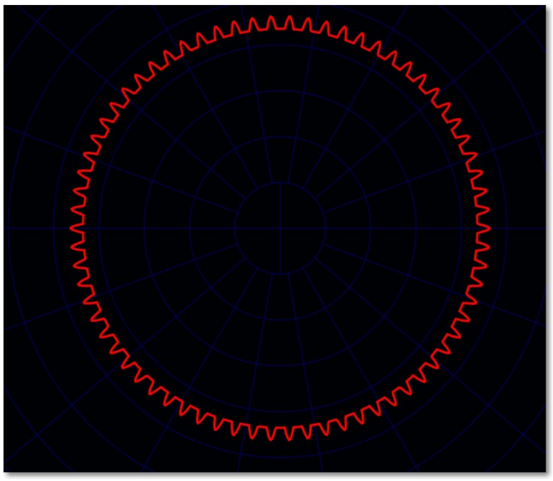
- The number of teeth in the gear
- The module value
- The pressure angle
- The weight has no effect at present as the diameter of the gear is solely dependent on the previous parameters.
- Phase
- Undercut – currently a fudge factor to get something like the correct tooth shape.



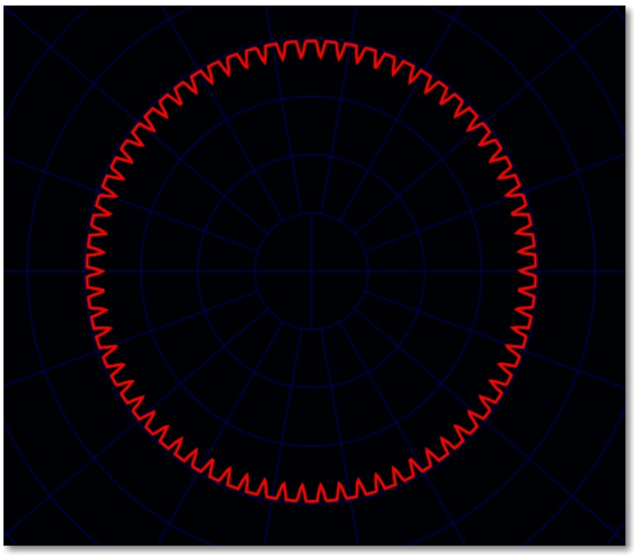
Gear-Mod1.2-N70-Pa15

Num Teeth	70
Module	1.20
Pressure Angle	15.0
Weight	2.0
Phase	-4.0
Undercut	1.00

Outside Diameter86.4
Root Diameter 81
Pitch Diameter 84



Gear wheel produced with parameters set as in image above. However setting a negative tool position does have an effect and it is to produce an internal gear wheel. That is one with the teeth pointing inward rather than outward. The path below was generated by setting the tool position to -80.



Remember that with all these rosettes a negative tool position simply means that on a real mechanical rose engine the tool is set to the opposite side of the barrel than the rubber.

Multiple Rosettes on a Barrel

When you add several rosettes to a barrel, the path traced is the combined outline of these rosettes.

Code Generation

The previous options were concerned with representing the tools path across the workpiece as a sequence of points. This section is concerned with transforming that sequence of points into a full G code program. The resulting program can then be saved to the disc for subsequent use.

This conversion from coordinates to gcode is performed using a collection of text Templates.

Templates

Templates are text strings that may contain replaceable symbols. A symbol is represented in the text by the following sequence of characters

......{Binding property} ....

Where **property** is some property value defined in the Data context of the code generator. When the code instantiator reads a symbol it is replaced by the corresponding property value. As an example the property CurrentPoint is defined in the code generator so that a template such as

X {Binding CurrentPoint.X} Y {CurrentPoint.Y}

Will result in a string value " **X 100 Y 50** " when evaluated with the current point being (100,50)

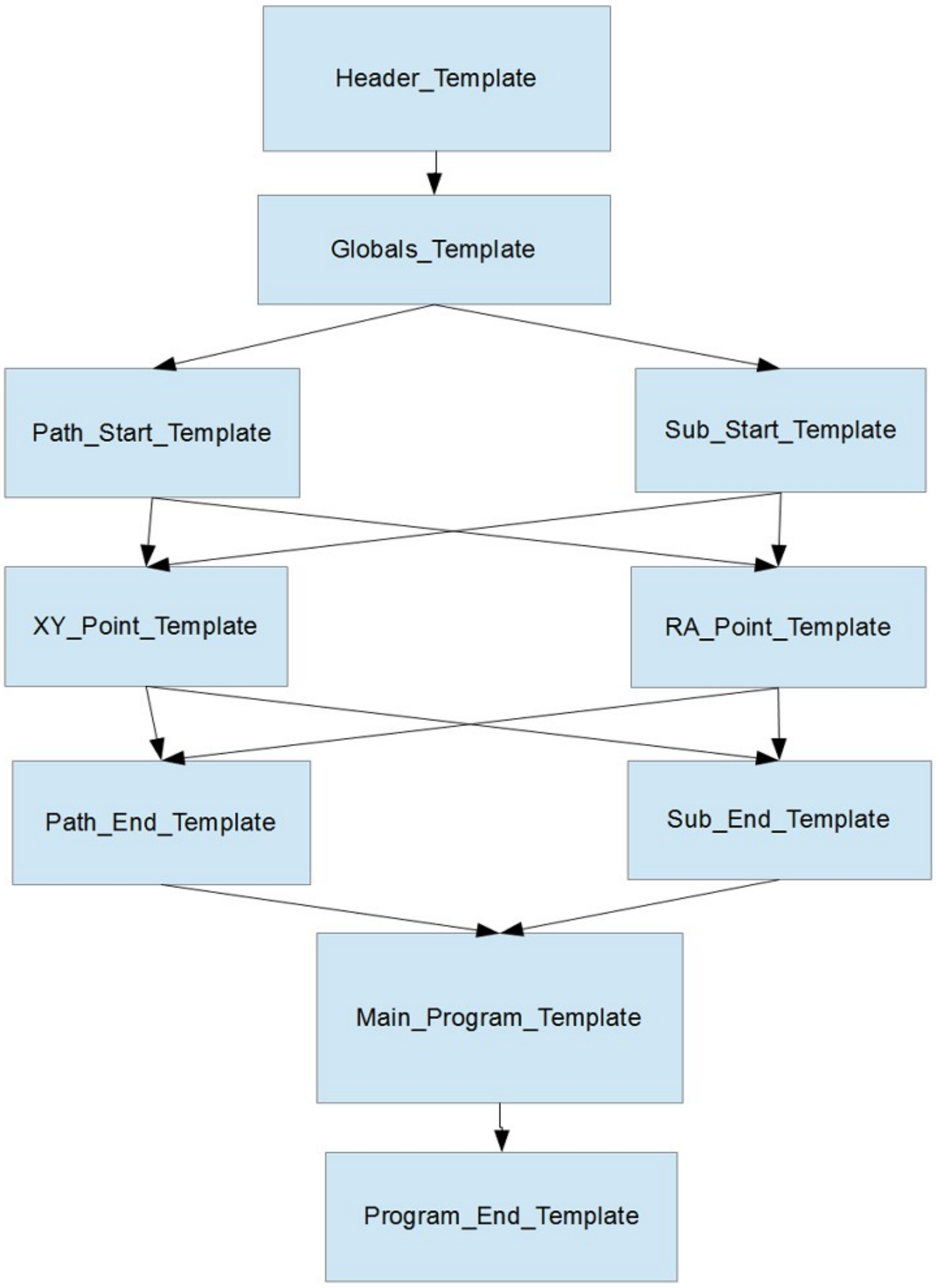
Properties Currently Supported

The properties supported include all the Application settings plus some others. These are

1. **UseRotary** - a flag set true if supporting a rotary table.
2. **UseAbsolute** - a flag set true if absolute positioning required.
3. **UseSub** - a flag set true if each gcode for path is to be generated as a subroutine.
4. **UseSingleFile** - a flag set true if only a single gcode file is to be generated.
5. **Now** - property giving current time and date.
6. **DP** - an integer giving number of decimal places values are rounded to.
7. **Feedrate** - a value giving the machine feed rate.
8. **CurrentPathIndex** - an integer representing the index of the current path.
9. **PatternName** - a string giving the name of the pattern used to produce the current path.
10. **SubPathName** - a string giving the auto generated subroutine / path name.
11. **PathCount** - an integer value giving the number of points in the current path.
12. **FirstPoint** - the first point on the current path.
13. **CurrentPoint** - the point on the current path being considered.
14. **Gcodes** - a table giving the vaues of G codes.
15. **Mcodes** - a table giving the values of M codes.
16. **SafeHeight** - value of cutter height for safe moves.
17. **CuttingDepth**- depth value of cutter when tracing path

G code program layout

G code programs usually follow a typical layout. This layout can be represented by a series of templates that are instantiated as the code generation process proceeds. OTWB represents this layout by the following sequence of templates



The template sequence starts with the HEADER_TEMPLATE which as its name suggests allows the user to display pertinent information about the program. Following this is the GLOBALS_TEMPLATE where you define your G code constants such as the safe working height or the cutting depth etc.

You can choose to opt for your paths to be generated as straight G code or encapsulated in a subroutine. The subroutine is necessary if you wish to store the path in a seperate file from the main program. So there is a corresponding choice between using the PATH_START_TEMPLATE or the SUB_START_TEMPLATE. I guess that strictly speaking only one template is necessary but for now I have chosen to separate these concerns at present . Both of these templates may include code to initialise matters before visiting each point on the path. Such initialisation code could for instance move the cutter to the safe height then male a rapid moveto the first point on the path and then finally making a linear move to lower the cutter to the current cutting depth.

Both of the two initialising templates mentioned above would be followed by repeated use of the next template which translates the points on the path into G code movements. The generation of the code for the path uses either the XY_POINT_TEMPLATE or the RA_POINT_TEMPLATE and this choice is dependent upon the type of Coordinates being used. Again I think that perhaps only one template is really necessary. The idea was that if you choose to use a rotary table or similar layout such as my rose engine. Then the best coordinates to use are Polar coordinates as they give you directly Radial (distance for tool to move horizontally) and an Angle (ammount of table rotation required) values. So for this use the RA_POINT_TEMPLATE. Otherwise for a standard mill use the XY_POINT_TEMPLATE.

At the end of the path some cleanup code is required such as moving the cutting tool back to its safe height etc. Either the PATH_END_TEMPLATE or the SUB_END_TEMPLATE should be used for this.

For very simple linear code the next template called the MAIN_PROGRAM_TEMPLATE is not necessary. However if you want to use subroutines and call them from within a loop then this is where you would place that code. Finally the PROGRAM_END_TEMPLATE allows you to add clean up code and terminate the program.

It is obvious from the discussion above that several templates may be superflouous if this is so just leave them empty or set their **Include** flag to false.

Default Template Values

Each of the templates mentioned above has a simple default value defined by OTWB. However remember that templates can be edited , saved and loaded so that you have the ability to specify numerous template examples. The default values are

HEADER_TEMPLATE

```
(    Gcode Program                     )
( BY:    OTWB                          )
( ON:    {Binding Now}                 )
( Flags:                               )
(   Rotary      : {Binding UseRotary}      )
(   Absolute    : {Binding UseAbsolute}    )
(   Subroutine  : {Binding UseSub}         )
(   Single File : {Binding UseSingleFile}  )
(                                      )
( Machine Settings                     )
(   Feed rate   : {Binding Feedrate}       )
(   Safe height : {Binding Safeheight}     )
(   Cut depth   : {Binding Cuttingdepth}    )
(   Accuracy    : {Binding DP}  dp          )
(   Num Paths   : {Binding PathCount}       )
(-------------------------------------------)
```

GLOBALS_TEMPLATE

```
<_safeheight> = {Binding Safeheight} ( Safe cutter height )
<_cutdepth> = {Binding CuttingDepth} ( Current cutter depth )
<_feedrate> = {Binding Feedrate }    ( Feed rate          )
```

PATH_START_TEMPLATE

```
(PATH {Binding CurrentPathName}  )
{Binding Gcodes.Absolute_Moves}
{Binding Gcodes.Rapid_Move} Z {Binding Safeheight}
{Binding Gcodes.Rapid_Move} X {Binding FirstPoint.X} Y {Binding FirstPoint.Y}
{Binding Gcodes.Linear_Move} Z {Binding Cuttingdepth} F {Binding Feedrate}
```

SUB_START_TEMPLATE

```
o<{Binding SubPathName}> sub
{Binding Gcodes.Absolute_Moves}
{Binding Gcodes.Rapid_Move} Z {Binding Safeheight}
{Binding Gcodes.Rapid_Move} X {Binding FirstPoint.X} Y {Binding FirstPoint.Y}
{Binding Gcodes.Linear_Move} Z {Binding CuttingDepth} F {Binding Feedrate}
```

XY_POINT_TEMPLATE

```
X{Binding CurrentPoint.X} Y{Binding CurrentPoint.Y}
```

RA_POINT_TEMPLATE

```
X{Binding CurrentPoint.Radius} C{Binding CurrentPoint.Angle}
```

PATH_END_TEMPLATE

```
{Binding Gcodes.Rapid_Move} Z {Binding Safeheight}
( END PATH {Binding CurrentPathName}  )
```

SUB_END_TEMPLATE

```
{Binding Gcodes.Rapid_Move} Z {Binding Safeheight}
o<{Binding SubPathName}> endsub
```

MAIN_PROGRAM_TEMPLATE

```
{Binding Gcodes.Rapid_Move} X{Binding FirstPoint.X}  Y{Binding FirstPoint.Y} F{Binding
Feedrate}
o<{Binding SubPathName}> call
```

PROGRAM_END_TEMPLATE

```
{Binding Mcodes.End_of_Program}
( ****** End of Program ****** )
```

Redundancies

It is becomming apparent to me that several of the binding properties I have included such as **Feedrate** and **Safeheight** could just be included as textual values in the GLOBALS_TEMPLATE for example

```
<_safeheight> = 5   ( Safe cutter height       )
<_cutdepth> = -1    ( Current cutter depth     )
<_feedrate> = 100   ( Feed rate                )
```

So I am currently undecided about keeping them as bindings.

Simple Example

The simplest program is just be a list of gcode movements one for each point on the tool path. So for example if a path comprised of just a few points say

$(0,0) \rightarrow (0,10) \rightarrow (10,10) \rightarrow (0,10) \rightarrow (0,0)$

Then a very simple gcode program would have one line of g code to move from point to point

```
[1]   g90            ( use absolute positioning  )
[2]   g0 Z 5         ( rapid move to a safe height )
[3]   g0 X0 Y0       ( rapid move to first point     )
[4]   g1 Z -1 F 20   ( linear move to cutting depth )
[5]   g1 X0 Y10      ( linear move to second point )
[6]   g1 X10 Y10     ( linear move to third point    )
[7]   g1 X0 Y10      ( linear move to fourth point )
[8]   g1 X0 Y0       ( linear move back to first point )
[9]   g0 Z 5         ( rapid move up to safe height )
```

```
[10]    M2                      ( end of program )
```

Note that there are some extra codes we have to include before we move along the path.

- The first line tells the controller that we are using absolute positions.
- The second line is included to raise the cutter to a safe position so that when we move to the first point on the path we do not hit anything.
- The third line rapidly moves to the first point on the path.
- The fourth line then feeds the cutter down to its cutting depth. Notice that with a linear move we have to specify a feed rate.

Lines 5 through 8 will move the cutter along the required path. Notice that we do not need to include a Z value again unless we want the cutting depth to change. Line 9 moves the cutter back to the safe height and line 10 stops the program.

A quick look should be sufficient to convince you that all paths no matter how many points fall into these three stages

1. Initialise and move to the first point
2. Loop through all the points on the path.
3. Final cleanup phase

To cover these stages the user only has to define these templates

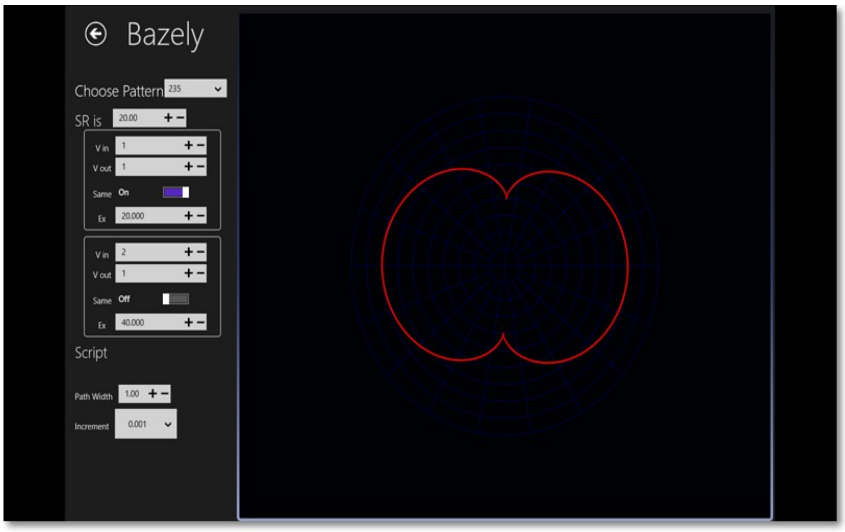
1. PATH_START_TEMPLATE
2. XY_POINT_TEMPLATE
3. PATH_END_TEMPLATE
4. PROGRAM_END_TEMPLATE

which in this simple case are covered by the default template values anyway. The only thing to do would be to ensure that the MAIN_PROGRAM_TEMPLATE was empty as it is unnecessary. Plus also there would be a Header which I have not shown.

A real example Using Subroutines

To opt to use subroutines set the UseSub flag on. The name of the subroutine will be automatically generated using the pattern name and the path index. So for Bazely pattern 235 which has one path the subroutine name would be **bazely#235_0 .** A subroutine is never executed until it is called so you need to make a call to this sub. This is done by the default MAIN_PROGRAM_TEMPLATE

If you start OTWB, go to the Bazely page and choose pattern 235.



Bazely

Choose Pattern: 235
SR is: 20.00

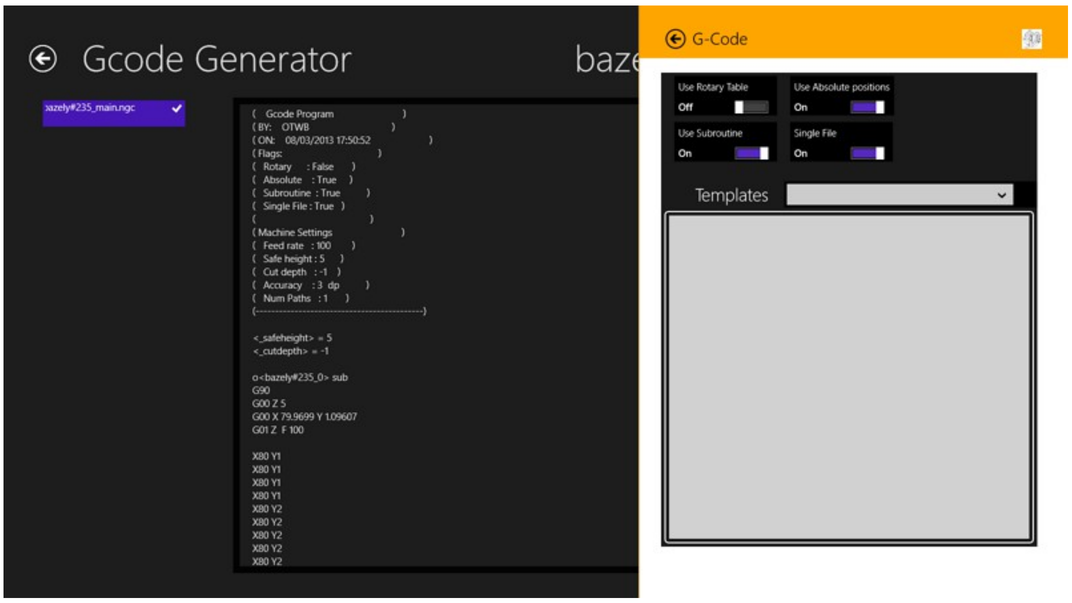
V in: 1
V out: 1
Same: On
Ex: 20.000

V in: 2
V out: 1
Same: Off
Ex: 40.000

Script

Path Width: 1.00
Increment: 0.001

Then go to the gode generation page. Make sure that in the settings the Use Subroutine and Single File switches are on as in the image below



Gcode Generator

bazely#235_main.ngc

```
(   Gcode Program                    )
( BY:   OTWB                    )
( ON:   08/03/2013 17:50:52                 )
( Flags:                    )
(  Rotary     : False      )
(  Absolute    : True     )
(  Subroutine  : True         )
(  Single File : True   )
(                         )
( Machine Settings                    )
(  Feed rate   : 100       )
(  Safe height : 5     )
(  Cut depth   : -1   )
(  Accuracy    : 3  dp         )
(  Num Paths   : 1      )
(------------------------------------------)

<_safeheight> = 5
<_cutdepth> = -1

o<bazely#235_0> sub
G90
G00 Z 5
G00 X 79.9699 Y 1.09607
G01 Z  F 100

X80 Y1
X80 Y1
X80 Y1
X80 Y1
X80 Y2
X80 Y2
X80 Y2
X80 Y2
X80 Y2
```

G-Code

Use Rotary Table: Off
Use Absolute positions: On
Use Subroutine: On
Single File: On

Templates

Then the code generated will be as in the image above. Because there are 6284 points on this path I have cut out most of them from the listing below which was created when the code was saved.

```
(    Gcode Program                           )
( BY:    OTWB                                )
( ON:    08/03/2013 18:12:27                 )
( Flags:                                     )
(   Rotary      : False                     )
(   Absolute    : True                      )
(   Subroutine  : True                      )
(   Single File : True                      )
(                                            )
( Machine Settings                           )
(   Feed rate   : 100                       )
(   Safe height : 5                         )
(   Cut depth   : -1                        )
(   Accuracy    : 3  dp                     )
(   Num Paths   : 1                         )
(-------------------------------------------)

<_safeheight> = 5
<_cutdepth> = -1

o<bazely#235_0> sub
G90
G00 Z 5
G00 X 79.9699 Y 1.09607
G01 Z-1  F 100
X79.97 Y1.096
X79.969 Y1.216
X79.967 Y1.336

...................

X79.972 Y0.714
X79.972 Y0.834
X79.971 Y0.954
X79.97 Y1.074
G00 Z 5
o<bazely#235_0> endsub

( ***** start of main program *****)

o<bazely#235_0> call
M2
( ***** end of gcode program *****)
```