Example files:

The following cubic bezier spline code examples for a 1" diameter circle will produce equivalent results in RiceCNC:

1.
```
G17 G02 X0 Y0 Z4 I1 J0
```

2.
```
use XY plane
helical cw X 0 Y 0 Z 4 centerX 1 centerY 0
```

3.
```
G01 X 0.0 0.4477153 1.0 Y 0.5522847 1.0 1.0 Z 1.0
feed X 1.5522847 2.0 2.0 Y 1.0 0.5522847 0.0 Z 2.0
feed to X 2.0 1.5522847 1.0 Y 0.5522847 -1.0 -1.0 Z 3.0
feed to X 0.4477153,0,0 Y 1,-0.5522847,-0.0 Z 4.0
```

Example 2 shows string substitution that improves readability but evaluates to the same codes as example 1. When executed, both examples 1 and 2 will be converted to cubic bezier spline specifications equivalent to example 3. Note also that a starting and ending velocity of 0.5522847 times the radius provides the closest approximation to a 90° circular arc. Using the expanded specification format of example 3, additional control points for P2 and P3 could be added to the Z axis to curve the Z motion as well as the X and Y motion. Thus any curved path in multi-dimensional space is possible.

Supplied Example Files:

Three example files are included in the application package and can be opened from the file menu.

The first file "Circular Path Example" demonstrates several techniques to cut a circular path including circular path to cubic Bezier spline conversion.

The second file "Drill Cycles" demonstrates the implemented drill cycle G-codes. Note that drill cycles could be implemented as program level subroutines but, because they are used frequently, it is more convenient to use the drill cycle G-codes.

The third file "Probe_Hole_ID" demonstrates a common function that is often implemented as a G-code. However, RiceCNC provides the capability to implement this and other, more complex, probing operations at program level and even to call the subroutine from a different program. To call a subroutine from a different program, the subroutine is label reference is preceded by the file name, with spaces replaced by underscores. The subroutine label must also be defined in the calling program. Control will pass to the local label if the referenced file is not open and compiled error free.

Circular Path Example

```
( This example includes three programs each demonstrating a different
  method to cut a 1" diameter circle (excluding cutter diameter)
  with equivalent results.

  The first program generates 361 points to cut the circle using
  short line segments.

  The second program uses G-code 02 and is recommended.

  The third program generates four Bezier splines to cut the circle
  Clearly you wouldn't do this for a simple circle on an orthogonal
  plane, but notice how easy it would be to change this program
  to cut an oval or add a bend in the Z axis.

  The fourth program uses polar coordinate mode to generate 361 points
```

```
  around the circle

  First program
  Adapted from a TurboCNC example, this program tests the ability to
  follow a path comprised of many coordinate points rather than a few
  Bezier splines )

call :get_parameters
#steps    = 360                    ; # of steps for a full circle
#delta    = [ 360 / #steps ]       ; angular increment in degrees

#theta    = 0                      ; current angle
call :calc_x_y
rapid to X #x_position Y #y_position ; traverse to start position
optional stop
optionally spindle on                    ; enable optionals run spindle
pause
feed to Z _cutting_plane           ; lower Z-axis to make cut
pause
call :cut_circle
jump to :program_end


get_parameters:
ask _diameter                  ; Diameter of the circle (inches)
if _diameter LE 0 end
#radius     = [ _diameter / 2 ]

ask _cutting_plane             ; Z for cutting plane
ask _release_plane             ; Z for release plane
end if _release_plane LE _cutting_plane

ask _center_X                  ; X-coord of center (inches)
ask _center_Y                ; Y-Coord of center (inches)

ask _feedrate                  ; feedrate in ipm (0 is rapid)
end if _feedrate LT 0
set feedrate to _feedrate

use XY plane
rapid to Z _release_plane     ; retract Z-Axis
return

; *** Subroutine to cut incremental circle ***
cut_circle: #theta += #delta           ; Increment the angle
call :calc_x_y
feed to X #x_position Y #y_position  ; make the cut
if #theta < 360 jump to :cut_circle  ; loop until complete
return

calc_x_y: ; clockwise circle coordinates
say #theta
#x_position = [ _center_X + #radius * COS( #theta ) ]
#y_position = [ _center_Y - #radius * SIN( #theta ) ]
return


( Second program
  For comparison, this is an equivalent program using the helical
  motion G code 2. To run this program, compile, then select an
  entire line of this block of comments, then select step to set the
  program index, then select run. )

call :get_parameters
```

```
#theta     = 0
call :calc_x_y
rapid to X #x_position Y #y_position ; start position
option spindle run             ; enable optionals run spindle
pause
feed to Z _cutting_plane      ; lower Z-axis to start cut
pause
; next command converts to four 90° bezier splines
circle cw centerX _center_X centerY _center_Y
jump to :program_end


( Third program
  This is an equivalent third example program uses the expanded
  Bezier curve specification to cut the circle.
  To run this program, compile, then select an entire line,
  then select step to set the program index, then select run. )

( Uncomment to test rotation in an orthogonal axis )
cancel rotation
( Assume machining a long slope, e.g., Hammond 1599, console box.
  We want to first rotate in XY 90° effectively exchanging X and Y,
  then rotate in YZ plane for sloped surface.
  We specify the rotations in reverse order because the last
  concatenated affine transformation will have precedence and
  effectively be executed first.

use YZ plane
rotate angle 5.87 Y _center_Y Z 0.0
use XY plane
rotate angle 90 X _center_X Y _center_Y

  equivalent code using I,J,K vector
rotate angle 5.87 Y _center_Y Z 0.0 I1
rotate angle 90 X _center_X Y _center_Y K1 )

call :get_parameters

#Xp1       = [ _center_X + #radius ]
#Yp1       = _center_Y
rapid to X #Xp1 Y #Yp1              ; start position
opt spindle on                      ; enable optionals run spindle
pause
feed to Z _cutting_plane          ; lower Z-axis to start cut
pause

#v         = [ #radius * 0.5522847 ]  ; value for minimum error of 0.02%

#Xp2       = #Xp1
#Xp3       = [ _center_X + #v ]
#Xp4       = _center_X

#Yp2       = [ _center_Y - #v ]
#Yp3       = [ _center_Y - #radius ]
#Yp4       = #Yp3

X #Xp2 #Xp3 #Xp4  Y #Yp2 #Yp3 #Yp4

#Xp2       = [ _center_X - #v ]
#Xp3       = [ _center_X - #radius ]
#Xp4       = #Xp3

#Yp2       = [ _center_Y - #radius ]
```

```
#Yp3      = [ _center_Y - #v ]
#Yp4      = _center_X

X #Xp2 #Xp3 #Xp4  Y #Yp2 #Yp3 #Yp4

#Xp2      = [ _center_X - #radius ]
#Xp3      = [ _center_X - #v ]
#Xp4      = _center_X

#Yp2      = [ _center_Y + #v ]
#Yp3      = [ _center_Y + #radius ]
#Yp4      = #Yp3

X #Xp2 #Xp3 #Xp4  Y #Yp2 #Yp3 #Yp4

#Xp2      = [ _center_X + #v ]
#Xp3      = [ _center_X + #radius ]
#Xp4      = #Xp3

#Yp2      = [ _center_Y + #radius ]
#Yp3      = [ _center_Y + #v ]
#Yp4      = _center_X

X #Xp2 #Xp3 #Xp4  Y #Yp2 #Yp3 #Yp4

program_end:
pause                       ; wait for motors to stop
spindle off
rapid to Z _release_plane ; retract Z-Axis
pause                       ; Wait for Z before displaying end
cancel rotation
end                         ; End of program stop


( Fourth program
  This equivalent forth example uses polar coordinate
  mode to cut the circle. To run this program,
  compile, then select an entire line in this comment block,
  then select step to set the program index, then select run. )

call :get_parameters

set polar origin to X _center_X Y _center_Y    ; start position
#steps    = 360                    ; # of steps for a full circle
#delta    = [ 360 / #steps ]       ; angular increment in degrees

#theta    = 360                      ; current angle
rapid to X #radius Y 0               ; X is radius, Y is angle
opt spindle on                       ; enable optionals run spindle
pause
feed to Z _cutting_plane             ; lower Z-axis to start cut
pause
call :cut_circle_polar
cancel polar
jump to :program_end

cut_circle_polar: #theta -= #delta  ; Increment the angle
say #theta
Y #theta                                    ; make the cut
if #theta >= 0 jump to :cut_circle_polar  ; loop until complete
return


; This example is not working with this release. It will be fixed in a future release.
```

```
( Fifth program
  This equivalent fifth example program uses coordinate rotation
  to cut the circle. Note that his example cannot be used in
  combination with axis scaling to cut an oval because the axis
  scaling will rotate along with the object coordinate.
  To run this program, compile, then select an entire line in this
  comment block, then select step to set the program index, then
  select run. )

cancel rotation
call :get_parameters

#steps    = 360                    ; # of steps for a full circle
#delta    = [ 360 / #steps ]       ; angular increment in degrees

#theta    = 0                      ; current angle
#offset     = [ _center_X + #radius ]
rapid to X #offset Y 0              ; X is radius, Y is angle
opt spindle on                      ; enable optionals run spindle
pause
feed to Z _cutting_plane            ; lower Z-axis to start cut
pause
call :cut_circle_rotation
jump to :program_end

cut_circle_rotation: #theta += #delta     ; Increment the angle
say #theta
rotate angle #delta X _center_X Y _center_Y
X #offset Y 0  ; make the cut
if #theta < 360 jump to :cut_circle_rotation  ; loop until complete
return
```

Drill Cycles

```
( Drill code example file )

cancel fixed cycles ; G code 80

use X-Y Plane   ; Drilling axis is Z

( G code 73 - Chip clearing drill cycle
  Only a partial return to release plane between pecks
  rapid to release plane
  set depth to release plane )

( LOOP
  exit loop if Z >= depth
  set target_depth to max( [ depth - peck ], z )
  feed to target_depth
  rapid to depth - chip clearing
  set depth to target_depth
  rapid to depth - return to hole bottom
  repeat LOOP )

; rapid to release plane

chip breaking drill X 0 Y 0 Z 0 release 1.0 peck 0.1 f 1
X 0 y 1 ; drill again at this position


( G code 81 - continuous drill, rapid out
  rapid to release plane
  continuous feed to Z
  rapid to release plane )
drill X 0.1 Y 0 Z 0 release 1.0
```

```
( G code 82 - continuous drill, dwell, rapid out
  rapid to release plane
  continuous feed to Z
  dwell
  rapid to release plane )
drill, dwell X 0.2 Y 0 Z 0 release 1.0 duration 1


( G code 83 - Chip clearing with
  full return to release plane between pecks
  rapid to release plane
  set depth to release plane )

( LOOP
  exit loop if Z >= depth
  set target_depth to max( [ depth - peck ], z )
  feed to target_depth
  rapid to release plane
  set depth to target_depth
  rapid to depth - return to hole bottom
  repeat LOOP )

; rapid to release plane

chip clearing drill X 0.3 Y 0 Z 0 release 1.0 peck 0.1 f 1


( G code 85 - continuous drill, feed out
  rapid to release plane
  continuous feed to Z
  feed to release plane )
drill, feed out X 0.5 Y 0 Z 0 release 1.0


( G code 86 - continuous drill, dwell, rapid out with spindle stopped
  rapid to release plane
  spindle on
  continuous feed to Z
  dwell
  spindle off
  rapid to release plane
  spindle on )
drill, dwell, spindle off, out X 0.6 Y 0 Z 0 release 1.0 duration 1


( G code 89 - continuous drill, dwell, feed out
  rapid to release plane
  continuous feed to Z
  dwell
  feed to release plane )
drill, dwell, feed out X 0.9 Y 0 Z 0 release 1.0 duration 1


; Now test another plane
use Z-X Plane   ; Drilling axis is Y

chip breaking drill X 0 Y 0 Z 0 release 1.0 peck 0.1 f 1
X 0 Z 1 ; drill again at this position

drill X 0.1 Y 0 Z 0 release 1.0

drill, dwell X 0.2 Y 0 Z 0 release 1.0 duration 1
```

```
chip clearing drill X 0.3 Y 0 Z 0 release 1.0 peck 0.1 f 1

drill, feed out X 0.5 Y 0 Z 0 release 1.0

drill, dwell, spindle off, out X 0.6 Y 0 Z 0 release 1.0 duration 1

drill, dwell, feed out X 0.9 Y 0 Z 0 release 1.0 duration 1


; Now test another plane
use Y-Z Plane   ; Drilling axis is X

chip breaking drill X 1 Y 0 Z 0 release 0.0 peck 0.1 f 1
Y 0 Z 1 ; drill again at this position

drill X 1 Y 0.1 Z 0 release 0

drill, dwell X 1 Y 0.2 Z 0 release 0 duration 1

chip clearing drill X 1 Y 0.3 Z 0 release 0 peck 0.1 f 1

drill, feed out X 1 Y 0.5 Z 0 release 0

drill, dwell, spindle off, out X 1 Y 0.6 Z 0 release 0 duration 1

drill, dwell, feed out X 1 Y 0.9 Z 0 release 0 duration 1

end
```

Probe_Hole_ID

```
; Example to probe a hole inside diameter to find the exact center

( Many software packages implement this common function as a G-code.
  RiceCNC does not implement this common function as a G-code, rather
  it provides the capability to perform this function and similar
  functions at program level. )

( To call this subroutine from another program, you would add the
  filename, with spaces replaced by underscore, before the label. )

; call Probe_Hole_ID:probe_hole_id

set feedrate to 1 ; initialization if not called from another program
_diameter = 1

probe_hole_id:
; calling parameters _diameter
; Feedrate must be set for probing moves

; set program offsets pushes the current coordinate offsets on a stack
; set program offsets X0 Y0 Z0 ; shift current position to origin

pause     ; Insure that all motors are idle
#x0 = __pos_X
#y0 = __pos_Y
#z0 = __pos_Z

if __circular_plane EQ 0 call :center_in_XY
if __circular_plane EQ 1 call :center_in_ZX
if __circular_plane EQ 2 call :center_in_YZ
```

```
; pop program offsets          ; restore prior offsets
return                        ; now at hole center

center_in_XY:
call :center_X
call :center_Y
call :center_X ; repeat X search for greater accuracy
return

center_in_ZX:
call :center_X
call :center_Z
call :center_X ; repeat X search for greater accuracy
return

center_in_YZ:
call :center_Y
call :center_Z
call :center_Y ; repeat Y search for greater accuracy
return

center_X:
probe to X [ #x0 + _diameter ]
#x1 = __pos_X
say #x1
end if __probe_stop eq -1 ; Probe active at start
end if __probe_stop eq 1  ; Probe did not contact workpiece
rapid to X #x0
probe to X [ #x0 - _diameter ]
#x2 = __pos_X
say #x2
end if __probe_stop eq -1 ; Probe active at start
end if __probe_stop eq 1  ; Probe did not contact workpiece
#x0 = [ ( #x1 + #x2 ) / 2 ]
rapid to X #x0
return

center_Y:
probe to Y [ #y0 + _diameter ]
#y1 = __pos_Y
say #y1
end if __probe_stop eq -1 ; Probe active at start
end if __probe_stop eq 1  ; Probe did not contact workpiece
rapid to Y #y0
probe to Y [ #y0 - _diameter ]
#y2 = __pos_Y
say #y2
end if __probe_stop eq -1 ; Probe active at start
end if __probe_stop eq 1  ; Probe did not contact workpiece
#y0 = [ ( #y1 + #y2 ) / 2 ]
rapid to Y #y0
return

center_Z:
probe to Z [ #z0 + _diameter ]
#z1 = __pos_Z
say #z1
end if __probe_stop eq -1 ; Probe active at start
end if __probe_stop eq 1  ; Probe did not contact workpiece
rapid to Z #z0
probe to Z [ #z0 - _diameter ]
#z2 = __pos_Z
say #z2
end if __probe_stop eq -1 ; Probe active at start
```

```
end if __probe_stop eq 1  ; Probe did not contact workpiece
#z0 = [ ( #z1 + #z2 ) / 2 ]
rapid to Z #z0
return
```