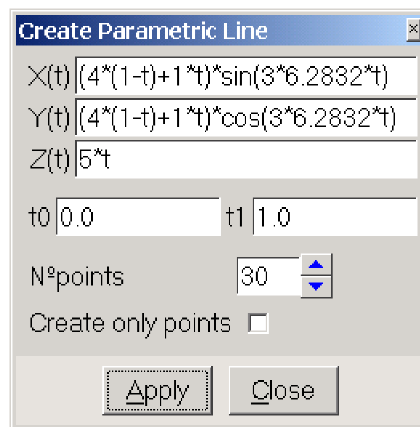
Advanced Customization

EXTENDING GID WITH TCL/TK

The objective of this case study is implementing a new tool in GiD to create parametric curves; the user introduces the general formula of these curves into a new window:



Parametric curves window

The required input datas are the mathematical formulas of the coordinates X(t), Y(t), Z(t), where 't' is the parameter of the curve, and its value belongs at the interval [t0-cT1]

The curve is created by approximation. That curve is a NURB (Non Uniform Rational B-Spline) which is created with N points. In GiD these kind of curves are created with degree 3 (cubical).

GiD uses Tcl/Tk to create its graphical interface; Tcl/Tk is a simple textual programming language.

This example works with GiD version 6.2.0b or later and Tcl/Tk 8.3.3.

TABLE OF CONTENTS

CREATING PARAMETRIC CURVES **22**

1. CREATING A MENU 22
2. CREATING A WINDOW WITH TK 24
3. CREATING THE WINDOW 25
4. CREATING THE PARAMETRIC CURVE 29
5. EXAMPLES 34
6. ADDITIONAL INFORMATION ABOUT TCL/TK 35

CREATING PARAMETRIC CURVES

1. Creating a menu

The procedure that creates the new window will be called **ParametricLine**. Before implementing the code of this procedure, we will modify the GiD menus, so that it is possible to call **ParametricLine** from a menu option.

The standard menus and windows of GiD could be directly modified editing the files contained in the \scripts directory, but this is not a good solution because these files often change depending on the GiD version.

To customize GiD it's recommended to create a "problem type", which will perform all the necessary modifications.

A *problem type* is a set of files configured by GiD so that the program can prepare data to be analyzed. The problem type can also modify menus and create new windows.

GiD offers some functions to change the **GiD** menus. With these functions it is possible to add new menus or to change the existing ones. These functions are: CreateMenu, InsertMenuOption, RemoveMenuOption y UpdateMenu.

For example, if our problem type is called Test1, we'll have to create a new directory "Test1.gid" inside the \problemtypes directory. Inside "Test1.gid" it has to be a file called "Test1.tcl" which will modify the GiD menu and create a window.

The structure of Test1.tcl is composed by some procedures; one of them can be InitGIDProject. InitGIDProject is called when the problem type "Test1" is selected in GiD. It receives the `dir` argument, which is the absolute path to the test1.gid directory.

You can find more information about TCL-TK and GiD in the GiD Reference Manual in the “TCL-TK extension” chapter.

InitGidProject adds a new menu called "Tools", with the option “Parametric Line”, which will call our procedure ParametricLine.

```
proc InitGIDProject { dir } {
  #WarnWinText prints a message in a window without stopping execution
  #WarnWin prints a message in a dialog (stops exectuion)
  #WarnWinText "InitGIDProject $dir"
  CreateMenu "Tools" "PRE"
  InsertMenuOption "Tools" "Parametric Line" 0 "ParametricLine" "PRE"
  UpdateMenus
}
```

ParametricLine will also be implemented in the file Test1.tcl.

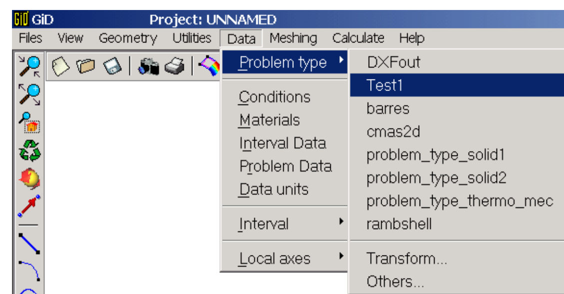
NOTE: ParametricLine can be implemented in an other file, but the file tclindex has to be modified:

For example, if ParametricLine was in a file called Parametric.tcl, it would be necessary to add the following line in tclindex:

set auto_index(ParametricLine) [list source [file join $dir Parametric.tcl]]

It's recommended to create the tclindex file automaticaly using the auto_mkindex utility. More information about tclindex in any TCL manual.

After selecting the Test1 problem type in GiD:



A new menu "Tools" appears between “Calculate” and “Help”.

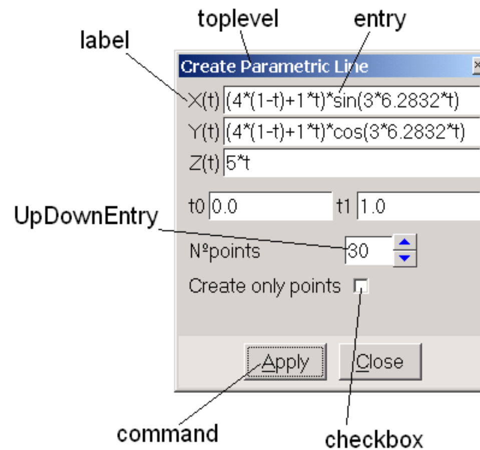
2. Creating a window with Tk

First, we need to create the different objects of the window: buttons, labels, text entries, etc.

The procedure which creates the window will be called ‘ParametricLine’.

The graphic objects (“Tk widgets”) that will be used are: (note that Tcl/Tk is case sensitive)

- toplevel: The toplevel command creates a new toplevel widget (creates new windows)
- frame: A frame is a simple widget. Its primary purpose is to act as a spacer or container for complex window layouts.
- label: A label is a widget that displays a textual string, bitmap or image
- entry: An entry is a widget that displays a one-line text string and allows that string to be edited
- checkbutton: A checkbutton is a widget that displays a textual string, bitmap or image and a square called an indicator. In addition, checkbuttons can be selected. If a checkbutton is selected then the indicator is normally drawn with a selected appearance.
- button: A button is a widget that displays a textual string, bitmap or image. The user can invoke the button by pressing mouse button 1 with the cursor over the button.
- UpDownEntry: this is not a Tcl standard widget, it's a derivated widget composed by an entry and two buttons to increment/decrement the value of that entry.

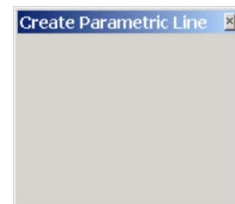


3. Creating the window

First, we will create a window with the label "X(t)" and its entry. In that entry the user will enter the X(t) formula.

In the file Test1.tcl, we write the following code:

```
proc ParametricLine { } {
    global ParametricPriv
    set w .gid.parametric
    #creating the main window
    InitWindow $w "Create Parametric Line" ParametricWindowGeom
    #creating the widgets
    frame $w.frmEquations
    label $w.frmEquations.lx -text "X(t)"
    entry $w.frmEquations.ex -relief sunken -width 15 -textvariable ParametricPriv(xt)
}
```



The InitWindow command is implemented in GiD and it creates a new window (similar to toplevel). InitWindow is included in file tclfileP.tcl

The other tcl commands used are:

global: declares the given varname's to be global variables rather than local ones.

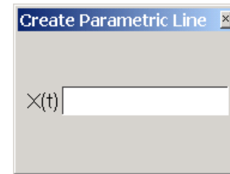
set: writes a value to a variable. In this example w is set to ".gid.parametric".

To get the value of a variable a $ has to be put before the name of the variable.

Symbol # is used to comment a line.

We have named the new window as .gid.parametric , and its title is “Create Parametric Line”. (arguments of InitWindow)

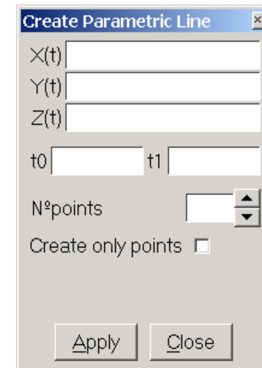
We have created the label and entry widgets but they are not visible yet; to put these widgets in the window it's necessary to use a “Window manager” command. In this example we will use the "grid" command.




```
proc ParametricLine {} {

   global ParametricPriv
    set w .gid.parametric
    #creating the main window
    InitWindow $w "Create Parametric Line" ParametricWindowGeom
    #creating the widgets
    frame $w.frmEquations
    label $w.frmEquations.lx -text "X(t)"
    entry $w.frmEquations.ex -relief sunken -width 15 -textvariable ParametricPriv(xt)
    #packing the widgets
    grid $w.frmEquations -padx 5 -pady 5 -sticky ew
    grid $w.frmEquations.lx $w.frmEquations.ex -sticky e
    grid conf $w.frmEquations.ex -sticky ew
}
```

Arguments "–padx 5 -pady 5" add an additional space around the widgets, and "–sticky ew" means that our widget will always stretched to fill the entire width of its cavity. ("ew" means east-west)

The rest of widgets are created and positioned in the window:




```
proc ParametricLine { } {
    global ParametricPriv
    set w .gid.parametric
    #creating the main window
    InitWindow $w "Create Parametric Line" ParametricWindowGeom
    #creating the widgets
    frame $w.frmEquations
    label $w.frmEquations.lx -text "X(t)"
    entry $w.frmEquations.ex -relief sunken -width 15 -textvariable ParametricPriv(xt)
    label $w.frmEquations.ly -text "Y(t)"
    entry $w.frmEquations.ey -relief sunken -width 15 -textvariable ParametricPriv(yt)
    label $w.frmEquations.lz -text "Z(t)"
    entry $w.frmEquations.ez -relief sunken -width 15 -textvariable ParametricPriv(zt)
    frame $w.frmParams
    label $w.frmParams.lt0 -text "t0"
    entry $w.frmParams.et0 -relief sunken -width 6 -textvariable ParametricPriv(t0)
    label $w.frmParams.lt1 -text "t1"
    entry $w.frmParams.et1 -relief sunken -width 6 -textvariable ParametricPriv(t1)
    frame $w.frmDivisions
    label $w.frmDivisions.lnpt -text "Nºpoints"
    UpDownEntry $w.frmDivisions.udnpt -variable ParametricPriv(NumberPointsT)\
                -entry-options "-relief sunken -width 4" \
                -up-command "IncrementNPT" -down-command "DecrementNPT"
    label $w.frmDivisions.lOnlyPoints -text "Create only points"
    checkbutton $w.frmDivisions.chkOnlyPoints -variable ParametricPriv(OnlyPoints)
    frame $w.frmButtons
    button $w.frmButtons.btnApply -text "Apply" \
                -command "CreateParametricLine $w" \
                -underline 0 -width 6
    button $w.frmButtons.btnclose -text "Close" \
                -command "destroy $w" \
                -underline 0 -width 6
    #packing the widgets
    grid $w.frmEquations -padx 5 -pady 5 -sticky ew
    grid $w.frmEquations.lx $w.frmEquations.ex -sticky e
    grid $w.frmEquations.ly $w.frmEquations.ey -sticky e
    grid $w.frmEquations.lz $w.frmEquations.ez -sticky e
    grid conf $w.frmEquations.ex $w.frmEquations.ey $w.frmEquations.ez -sticky ew
    grid columnconf $w.frmEquations 1 -weight 1
    grid $w.frmParams -padx 5 -pady 5 -sticky ew
    grid $w.frmParams.lt0 $w.frmParams.et0 $w.frmParams.lt1 $w.frmParams.et1 -sticky e
    grid conf $w.frmParams.et0  $w.frmParams.et1 -sticky ew
    grid columnconf $w.frmParams "1 3" -minsize 6 -weight 1
    grid $w.frmDivisions -padx 5 -pady 5 -sticky w
```

```
    grid $w.frmDivisions.lnpt $w.frmDivisions.udnpt -sticky w
    grid $w.frmDivisions.lOnlyPoints $w.frmDivisions.chkOnlyPoints -sticky w
    grid columnconf $w.frmDivisions 0 -weight 1
    grid $w.frmButtons -sticky ews -columnspan 7
    grid $w.frmButtons.btnApply $w.frmButtons.btnclose -padx 5 -pady 6
    grid columnconf $w "0" -weight 1
    grid rowconfigure $w "3" -weight 1
}
```

When the user clicks over the "Apply" button, procedure "CreateParametricLine" will be called. That procedure creates the curve using GiD commands.

"CreateParametricLine" receives an argument, the window's name (.gid.parametric, so, variable $w).

The rest of variables are stored in "ParametricPriv", a global variable which can be seen as a vector with all the values needed to create the curve: ParametricPriv(xt), ParametricPriv(NumberPointsT), etc.

The "UpDownEntry" widget is linked with the "ParametricPriv(NumberPointsT)" variable; so, when pressing "up" and "down" the values of that variable should change one unit up or down, but with a value always bigger than 1. When pressing "up" and "down" buttons, "IncrementNPT" y "DecrementNPT" procedures are called.

```
proc IncrementNPT { } {
    global ParametricPriv
    set ParametricPriv(NumberPointsT) [expr $ParametricPriv(NumberPointsT)+1]
}

proc DecrementNPT { } {
    global ParametricPriv
    # number of points cannot be smaller than 2
    if { $ParametricPriv(NumberPointsT) > 2} {
            set ParametricPriv(NumberPointsT) [expr $ParametricPriv(NumberPointsT)-1]
    } else {
            bell
    }
}
```

The "if-else" sentence checks if "$ParametricPriv(NumberPoints)" is bigger than 2; if false, the computer beeps. (bell)

Command "expr" evaluates the result as a Tcl expression, and returns the value.

4. Creating the parametric curve

The user will provide the data necessary to construct the parametric curve, which will be created in the "CreateParametricLine" procedure.

```
proc CreateParametricLine { w } {
    global ParametricPriv
    set fx $ParametricPriv(xt)
    set fy $ParametricPriv(yt)
    set fz $ParametricPriv(zt)
    # changing t byr $t in the formulas
    foreach {f} {fx fy fz} {
        regsub -all -nocase {\mt\M} [set $f] {$t} $f
    }
    set a $ParametricPriv(t0)
    set b [expr  ($ParametricPriv(t1)-$ParametricPriv(t0))/($ParametricPriv(NumberPointsT)-1.0)]
    set oldvalue [.central.s info variables CreateAlwaysNewPoint]
    .central.s process "escape escape escape Utilities Variables CreateAlwaysNewPoint  1"
    if { $ParametricPriv(OnlyPoints) } {
        .central.s process "escape escape escape Geometry Create Point"
    } else {
        .central.s process "escape escape escape Geometry Create NurbsLine"
    }
    for { set i 0 } { $i < $ParametricPriv(NumberPointsT) } { incr i} {
        set t [expr $a+$i*$b]
        set x [expr $fx]
        set y [expr $fy]
        set z [expr $fz]
        .central.s process "$x $y $z"
    }
    .central.s process "escape"
    .central.s process "escape escape escape Utilities Variables CreateAlwaysNewPoint $oldvalue"
    .central.s process "escape escape escape"
    .central.s process "redraw"
}
```

The user uses variable "t" to describe the formulas; we need to change that "t" by "$t" to make it a real Tcl variable. We use the regsub (regular expresssions) command to make the substitution.

"foreach" makes a rundown of the three formulas fx, fy, fz.

In these procedure an specific GiD command is used: .central.s. This command has two different subcommands, **process** and **info**.

Command "process" it's used to execute GiD commands (any command used in the GiD command bar, can be used with "process"); command "info" is used to get information about GiD and its database. You can find more information about these commands in the GiD Reference Manual.

For example, “.central.s info variables CreateAlwaysNewPoint” gives the value of an internal GiD variable, which controls the creation of new points (if new points are joined with existing ones if they are very close)

We need to set the value of that variable to 1, to force the creation of new points, even if they are close to an existing point. We do that with the process command:

```
.central.s process “Utilities Variables CreateAlwaysNewPoint 1”
```

We use the following command to create a Nurbs line:

```
.central.s process “Geometry Create NurbsLine”
```

followed by the coordinates of the points:

```
.central.s process “$x $y $z”
```

After the curve is created, the initial value of "CreateAlwaysNewPoint" is restored.

To get the coordinates we make a loop using command "for", where t changes from t0 to t1, with the number of divisions wished (number of points-1). We get the values of t, x, y, z using the command "expr".

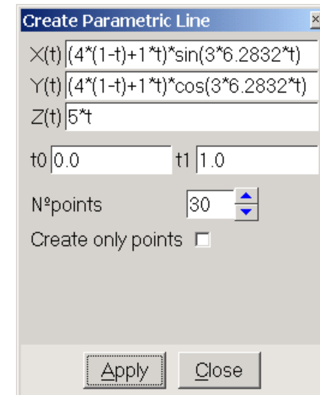
Finally, some other instructions have been aded to the program to initialize the value of the variables, improve the window design, avoid unnecessary redraws and manage possible user syntax errors.

Another GiD command has been used to help the user with the valid syntax of the formulas:

```
GidHelp “widget name” “Help text”
```

This command shows some text when the user clicks on the widget with the right mouse button.

The final code is:




```
proc ParametricLine { } {
    global ParametricPriv
    set w .gid.parametric
    set ParametricPriv(xt) "(4*(1-t)+1*t)*sin(3*6.2832*t)"
    set ParametricPriv(yt) "(4*(1-t)+1*t)*cos(3*6.2832*t)"
    set ParametricPriv(zt) "5*t"
    set ParametricPriv(NumberPointsT) 30
    set ParametricPriv(t0) 0.0
    set ParametricPriv(t1) 1.0
    set ParametricPriv(OnlyPoints) 0
    #creating the main window
    InitWindow $w "Create Parametric Line" ParametricWindowGeom
    #creating the widgets
    frame $w.frmEquations
    label $w.frmEquations.lx -text "X(t)"
    entry $w.frmEquations.ex -relief sunken -width 15 -textvariable ParametricPriv(xt)
    label $w.frmEquations.ly -text "Y(t)"
    entry $w.frmEquations.ey -relief sunken -width 15 -textvariable ParametricPriv(yt)
    label $w.frmEquations.lz -text "Z(t)"
    entry $w.frmEquations.ez -relief sunken -width 15 -textvariable ParametricPriv(zt)
    GidHelp "$w.frmEquations " \
        "Parametric equations, valid all tcl funcions: " \
        "+ - * / %% " \
        "abs cosh log sqrt acos double log10 srand asin exp pow" \
        "tan atan floor rand tanh atan2 fmod round ceil hypot sin" \
        "cos int sinh."
    frame $w.frmParams
    label $w.frmParams.lt0 -text "t0"
    entry $w.frmParams.et0 -relief sunken -width 6 -textvariable ParametricPriv(t0)
    label $w.frmParams.lt1 -text "t1"
    entry $w.frmParams.et1 -relief sunken -width 6 -textvariable ParametricPriv(t1)
    frame $w.frmDivisions
    label $w.frmDivisions.lnpt -text "Nºpoints"
    UpDownEntry $w.frmDivisions.udnpt -variable ParametricPriv(NumberPointsT)\
            -entry-options "-relief sunken -width 4" \
            -up-command "IncrementNPT" -down-command "DecrementNPT" \
            -button-options "-bd 1" \
            -up-color blue -down-color blue
    label $w.frmDivisions.lOnlyPoints -text "Create only points"
    checkbutton $w.frmDivisions.chkOnlyPoints -variable ParametricPriv(OnlyPoints)
    set def_back [$w cget -background]
    frame $w.frmButtons -bg [CCColorActivo $def_back]
    button $w.frmButtons.btnApply -text "Apply" \
            -command "CreateParametricLine $w" \
            -underline 0 -width 6
    button $w.frmButtons.btnclose -text "Close" \
            -command "destroy $w" \
            -underline 0 -width 6
    #packing the widgets
    grid $w.frmEquations -padx 5 -pady 5 -sticky ew
    grid $w.frmEquations.lx $w.frmEquations.ex -sticky e
    grid $w.frmEquations.ly $w.frmEquations.ey -sticky e
    grid $w.frmEquations.lz $w.frmEquations.ez -sticky e
    grid conf $w.frmEquations.ex $w.frmEquations.ey $w.frmEquations.ez -sticky ew
    grid columnconf $w.frmEquations 1 -weight 1
    grid $w.frmParams -padx 5 -pady 5 -sticky ew
    grid $w.frmParams.lt0 $w.frmParams.et0 $w.frmParams.lt1 $w.frmParams.et1 -sticky e
    grid conf $w.frmParams.et0  $w.frmParams.et1 -sticky ew
    grid columnconf $w.frmParams "1 3" -minsize 6 -weight 1
    grid $w.frmDivisions -padx 5 -pady 5 -sticky w
    grid $w.frmDivisions.lnpt $w.frmDivisions.udnpt -sticky w
    grid $w.frmDivisions.lOnlyPoints $w.frmDivisions.chkOnlyPoints -sticky w
    grid columnconf $w.frmDivisions 0 -weight 1
    grid $w.frmButtons -sticky ews -columnspan 7
    grid $w.frmButtons.btnApply $w.frmButtons.btnclose -padx 5 -pady 6
    grid columnconf $w "0" -weight 1
    grid rowconfigure $w "3" -weight 1
    focus $w.frmButtons.btnApply
    bind $w <Alt-c> "tkButtonInvoke $w.frmButtons.btnclose"
    bind $w <Escape> "tkButtonInvoke $w.frmButtons.btnclose"
    bind $w <Return> "tkButtonInvoke $w.frmButtons.btnApply"
}
```

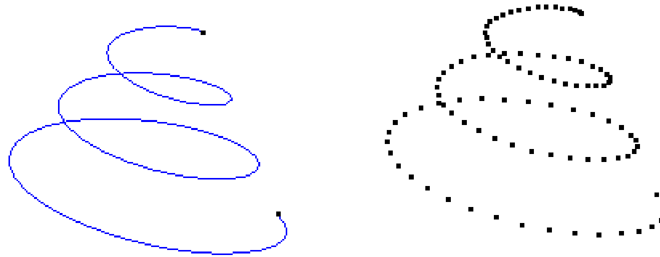
```
proc IncrementNPT { } {
   global ParametricPriv
   set ParametricPriv(NumberPointsT) [expr $ParametricPriv(NumberPointsT)+1]
}

proc DecrementNPT { } {
   global ParametricPriv
   if { $ParametricPriv(NumberPointsT) > 2} {
      set ParametricPriv(NumberPointsT) [expr $ParametricPriv(NumberPointsT)-1]
   } else {
      bell
   }
}

proc CreateParametricLine { w } {
   global ParametricPriv
   set fx $ParametricPriv(xt)
   set fy $ParametricPriv(yt)
   set fz $ParametricPriv(zt)
  #is data ok?
   if { ![string is double -strict $ParametricPriv(t0)] } {
         tk_messageBox -message "Error, t0 must be a real number" -type ok
         return
   }
   if { ![string is double -strict $ParametricPriv(t1)] } {
         tk_messageBox -message "Error, t1 must be a real number" -type ok
         return
   }
   if { ![string is integer -strict $ParametricPriv(NumberPointsT)] } {
         tk_messageBox -message "Error, Nº points must be a positive integer >=2" -type ok
         return
   }
   if { $ParametricPriv(NumberPointsT) < 2 } {
         tk_messageBox -message "Error, Nº points must be a positive integer >=2" -type ok
         return
   }
   # substitution of t by $t in the formulas
   foreach {f} {fx fy fz} {
         regsub -all -nocase {\mt\M} [set $f] {$t} $f
   }
   #do not redraw and change mouse cursor to a clock
   $w conf -cursor watch
   .central.s waitstate 1
   update
   .central.s disable graphics 1
   .central.s disable windows 1
   .central.s disable graphinput 1
   if { [ catch {
         set a $ParametricPriv(t0)
         set b [expr  ($ParametricPriv(t1)-$ParametricPriv(t0))/($ParametricPriv(NumberPointsT)-1.0)]
         set oldvalue [.central.s info variables CreateAlwaysNewPoint]
         .central.s process "escape escape escape Utilities Variables CreateAlwaysNewPoint  1"
         if { $ParametricPriv(OnlyPoints) } {
            .central.s process "escape escape escape Geometry Create Point"
         } else {
            .central.s process "escape escape escape Geometry Create NurbsLine"
         }
         for { set i 0 } { $i < $ParametricPriv(NumberPointsT) } { incr i} {
            set t [expr $a+$i*$b]
            set x [expr $fx]
            set y [expr $fy]
            set z [expr $fz]
            .central.s process "$x $y $z"
         }
         .central.s process "escape"
         .central.s process "escape escape escape Utilities Variables \
                 CreateAlwaysNewPoint $oldvalue"
         .central.s process "escape escape escape"
   } failstring ] } {
         tk_messageBox -message \
                 "Error, there is an error in the mathematical expression ($failstring)" \
                 -type ok -parent $w
   }
```

```
    #redraw and set mouse cursor to normal
    .central.s disable graphics 0
    .central.s disable windows 0
    .central.s disable graphinput 0
    .central.s process "redraw"
    $w conf -cursor ""
    .central.s waitstate 0
}
```

5. Examples

First, we fill the formulas with the expression of a conic helix. That helix starts with radius R_0=4 and finishes with radio R_1=1, performing N=3 turns from t=0.0 to t=1.0, the height also changes from 0 to H=5.

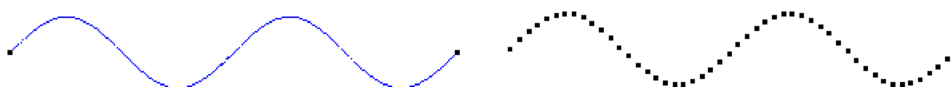
$$\begin{cases} x(t) = (R_0 \cdot (1-t) + R_1) \cdot \sin(N \cdot 2\pi \cdot t) \\ y(t) = (R_0 \cdot (1-t) + R_1) \cdot \cos(N \cdot 2\pi \cdot t) \\ z(t) = H \cdot t \end{cases}$$



Example of conic helix with a unique curve or with only points.

Next, is another example with a 2D curve y=f(x), using the sinus function, with parameter t changing from 0.0 to 4.0*pi

$$\begin{cases} x(t) = t \\ y(t) = \sin(t) \\ z(t) = 0 \end{cases}$$



Example of sinusoidal function

6. Additional information about Tcl/Tk

There are a lot of web pages with information, tutorials, etc. about tcl/tk. We recommend:

http://tcl.activestate.com

You can also find an interesting beginers tutorial of Tcl/Tk in spanish in:

http://www.etsimo.uniovi.es/tcl/tutorial

To find more information about GiD (manuals, tutorials, etc.) please go to:

http://gid.cimne.upc.es/support