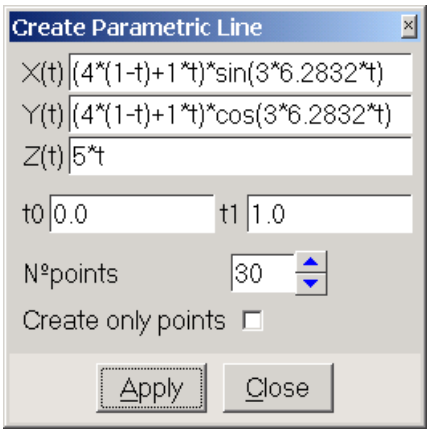
Personalización avanzada

EXTENSIÓN DE GID MEDIANTE SCRIPTS EN LENGUAJE TCL/TK

El objetivo de este ejemplo es añadir a GiD una nueva herramienta para crear curvas paramétricas, dichas curvas están descritas partir de su fórmula genérica introducida por el usuario.

Para ello, se crea una nueva ventana en la que introducir los datos necesarios, su aspecto será el siguiente:



Ventana de creación de curvas parametrizadas

Los datos de entrada requeridos son las fórmulas matemáticas de las coordenadas X(t), Y(t), Z(t) , donde 't' es el parámetro de la curva, y su valor pertenece al intervalo [t0-t1]

La curva que se crea es una aproximación. Se muestrean N puntos de paso, y se interpola por ellos una curva NURB (No Uniforme Racional B-Spline). En GiD esta curva interpolante se crea con grado 3 (cúbica).

El lenguaje que emplea GiD para su interface gráfica es Tcl/Tk, un lenguaje interpretado, fácilmente portable entre distintos sistemas y extensible.

Este tutorial está pensado para funcionar en GiD versión 6.2.0b o posteriores, la versión de Tcl/Tk empleada es la 8.3.3 (Versiones anteriores pueden no soportar algunas instrucciones de GiD o de Tcl)

Tabla de contenidos

CREACIÓN DE CURVAS PARAMETRIZADAS

1. Integración en un menú de GiD

Supongamos que el procedimiento que creará la ventana para crear curvas paramétricas se denominará ParametricLine. Antes de implementar el código de este procedimiento, modificaremos los menús de GiD, de modo que se pueda invocar fácilmente.

Los menús y ventanas estándar de GiD podrían modificarse editando directamente los ficheros contenidos en el directorio \scripts, pero esta posibilidad está completamente desaconsejada, ya que cada vez que se actualice el programa con una nueva versión, habría que volver a cambiar los nuevos ficheros.

El modo adecuado para personalizar GiD, es definir un “tipo de problema” (problem type), que al ser cargado realice todas las modificaciones necesarias.

Un “tipo de problema” consiste en un conjunto de ficheros de configuración para que el GiD conozca los campos de los datos relativos a los materiales, las condiciones de contorno a aplicar, el formato del fichero de entrada para el cálculo, etc.

Un “Tipo de problema” puede además modificar los menús estándar de GiD al ser cargado y definir nuevas ventanas.

GiD ofrece al desarrollador, algunas funciones tcl para facilitar la operación de incluir sus propios menús de usuario. Estas funciones son: CreateMenu, InsertMenuOption, RemoveMenuOption y UpdateMenu.

Por ejemplo, supongamos que nuestro tipo de problema se denomina Test1, debe crearse un directorio llamado Test1.gid en el directorio \problemtypes de GiD, y dentro del directorio Test1.gid, debemos crear un fichero Test1.tcl, donde se modifique el menú al ser cargado.

Desde GiD, se lanzan una serie de eventos tcl, que pueden ser interceptados en este fichero. Por ejemplo, al cargar un tipo de problema, GiD lanza el evento InitGIDProject, al que le pasa como parámetro la ruta del tipo de problema cargado.

En el manual de referencia de GiD, en el apartado “TCL-TK extension” puede consultarse los eventos tcl que genera.

Se puede aprovechar el evento InitGidProject para añadir un nuevo menú, llamado “Tools”, con un submenú llamado “Parametric Line” cuyo comando invoca a nuestro procedimiento ParametricLine

```
proc InitGIDProject { dir } {
  #WarnWinText imprime un mensaje en una ventana sin paralizar la ejecución
  #WarnWin imprime un mensaje en una ventana modal (detiene la ejecución)
  #WarnWinText "InitGIDProject $dir"
  CreateMenu "Tools" "PRE"
  InsertMenuOption "Tools" "Parametric Line" 0 "ParametricLine" "PRE"
  UpdateMenus
}
```

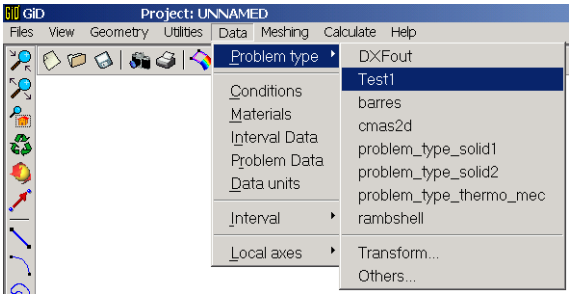
El procedimiento ParametricLine deberá estar dentro de este mismo fichero Test1.tcl, o bien escribiendo una entrada en el fichero tclindex, indicando a tcl donde se encuentra.

Por ejemplo, si ParametricLine estuviese contenido en el fichero Parametric.tcl, habría que añadir a tclindex una línea como la siguiente:

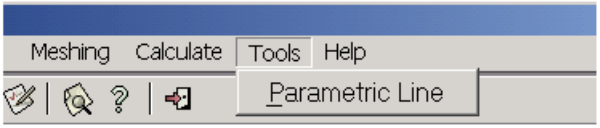
set auto_index(ParametricLine) [list source [file join $dir Parametric.tcl]]

Lo usual es crear el fichero tclindex automáticamente mediante la utilidad auto_mkindex, no es editarlo directamente. Puede consultarse más información acerca de tclindex en cualquier manual de tcl.

Al cargar en GiD el tipo de problema Test1:



Aparecerá un nuevo menú llamado “Tools“, situado entre los menús estándar “Calculate” y “help”, desde el que cualquier usuario sabrá desplegar nuestra ventana.



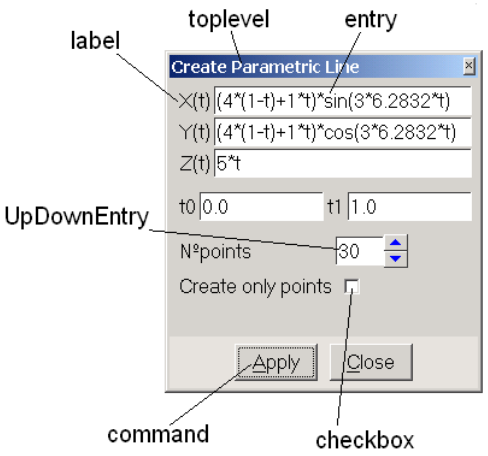
2. Componentes Tk de la ventana

En primer lugar se crearán los objetos que componen la ventana: botones, etiquetas, casillas de texto, etc.

El procedimiento que invocaremos para abrir nuestra ventana lo denominamos 'ParametricLine', dentro se incluye el código necesario para su creación.

Los objetos gráficos ("widgets") de Tk usados son los siguientes: (atención, en Tcl/Tk se distinguen mayúsculas de minúsculas)

- toplevel: ventana independiente
- frame: marco que contiene a otros controles
- label: etiqueta de texto decorativa
- entry: casilla de entrada de texto
- checkbutton: opción con valores activado / desactivado
- button: botón pulsable.
- UpDownEntry: (es un "witget" derivado, compuesto de un entry con botones para incrementar / decrementar su valor)

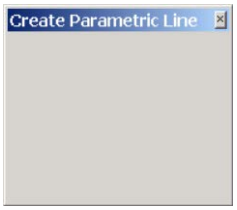


3. Construcción de la ventana

Para comenzar, se creará una ventana, solamente con un “label” X(t) y su “entry” para introducir la fórmula

Se añade para ello al fichero Test1.tcl el siguiente contenido:

```
proc ParametricLine { } {
   global ParametricPriv
   set w .gid.parametric
   #creación de la ventana principal
   InitWindow $w "Create Parametric Line" ParametricWindowGeom
   #creación de los witgets
   frame $w.frmEquations
   label $w.frmEquations.lx -text "X(t)"
   entry $w.frmEquations.ex -relief sunken -width 15 -textvariable ParametricPriv(xt)
}
```



En los scripts de tcl incluidos con GiD, se define el procedimiento InitWindow, para construir la ventana, que es de tipo toplevel (en concreto InitWindow está incluida en el fichero tclfileP.tcl)

Las instrucciones tcl restantes usadas son:

global: establece que una variable sea de ámbito global (por defecto son locales al procedimiento encerrado entre llaves)

set: establece el valor de una variable, en este caso w valdrá “.gid.parametric”

Todas las variables en Tcl son de tipo cadena de caracteres.

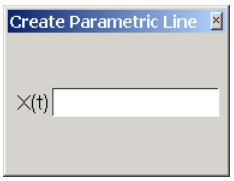
El valor de una variable se obtiene con el prefijo $ antes del nombre (sin espacios intermedios, los espacios separan instrucciones)

El símbolo # sirve para comentar una línea

La ventana se ha denominado .gid.parametric , y tiene por título “Create Parametric Line” (parámetros usados al invocar InitWindow)

Los “widgets” label y entry, se han creado, pero no se muestran hasta que no se “empaquetan” mediante el uso de un “Window manager”, en este caso se usará el comando grid para situar los “widgets” por filas y columnas.

```
proc ParametricLine {} {

  global ParametricPriv
   set w .gid.parametric
   #creación de la ventana principal
   InitWindow $w "Create Parametric Line" ParametricWindowGeom
   #creación de los witgets
   frame $w.frmEquations
   label $w.frmEquations.lx -text "X(t)"
   entry $w.frmEquations.ex -relief sunken -width 15 -textvariable ParametricPriv(xt)
   #empaquetamiento de los 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
}
```

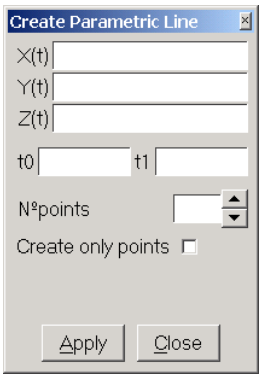


Los parámetros -padx 5, -pady 5 controlan el espacio adicional que rodea al "widget", y -sticky ew indica que al cambiar el tamaño de la ventana, el widget debe siempre llegar a los extremos este y oeste de su contenedor.

El comando grid conf permite modificar alguno de los parámetros por defecto o anteriormente establecidos.

Análogamente se construyen y muestran el resto de “widgets”:

```
proc ParametricLine { } {
   global ParametricPriv
   set w .gid.parametric
   #creación de la ventana principal
   InitWindow $w "Create Parametric Line" ParametricWindowGeom
   #creación de los witgets
   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
   #empaquetamiento de los 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
}
```

Al pulsar el botón Apply, se ejecutará el procedimiento CreateParametricLine que construye la curva mediante el uso de instrucciones propias de GiD.

Se le pasa a dicho procedimiento como parámetro el nombre de la ventana (.gid.parametric), que está contenido en la variable $w

El resto de parámetros necesarios, se guardan en la variable global ParametricPriv, que puede considerarse como una especie de vector que contiene ParametricPriv(xt), ParametricPriv(NumberPointsT), etc.

El seudo-widget de tipo UpDownEntry, está asociado a la variable ParametricPriv(NumberPointsT), al pulsar los botones adjuntos, se debe incrementar o decrementar esta variable en una unidad, impidiendo que el valor sea inferior a 2 puntos. Se han asociado a estos eventos las funciones IncrementNPT y DecrementNPT, que se definen del siguiente modo:

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

proc DecrementNPT { } {
    global ParametricPriv
    # no se permite un número de puntos de muestreo inferior a 2
    if { $ParametricPriv(NumberPointsT) > 2} {
            set ParametricPriv(NumberPointsT) [expr $ParametricPriv(NumberPointsT)-1]
    } else {
            bell
    }
}
```

La instrucción if-else ejecuta una sentencia si se cumple cierta condición, por ejemplo, si el número de puntos es menor o igual que dos suena una campana (bell) indicando el error.

El comando expr evalúa una expresión matemática (esta función será crucial para evaluar mas adelante las expresiones paramétricas de la curva)

4. Creación de la curva paramétrica

Mediante los “widgets” de la ventana, el usuario proporcionará los datos necesarios para construir la curva parametrizada, la cual se creará en el procedimiento CreateParametricLine.

```
proc CreateParametricLine { w } {
    global ParametricPriv
    set fx $ParametricPriv(xt)
    set fy $ParametricPriv(yt)
    set fz $ParametricPriv(zt)
    # sustitución de t por $t en las fórmulas
    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"
}
```

La variable de las fórmulas escritas por el usuario es t, debe sustituirse en la cadena de la fórmula este carácter por $t. La instrucción regsub (expresiones regulares) se encarga de esta operación.

foreach hace un bucle para pasar por las tres fórmulas fx, fy, fz

En este procedimiento se utilizan comando específicos de GiD, el objeto .central.s es un “widget” especial, que tiene dos comandos básicos: process e info

Mediante process se ejecutan comandos de GiD (cualquier orden escrita en la línea de comandos de GiD es válida para usar en process) , y mediante info se obtiene información de la base de datos de GiD. En el manual de referencia de GiD puede encontrarse información acerca de estos comandos.

Por ejemplo “.central.s info variables CreateAlwaysNewPoint” obtiene el valor de una variable interna de GiD, que controla la creación de puntos (para evitar superponer o no puntos muy próximos)

Establecemos el valor de esta variable a 1, para que se creen todos los puntos sin importar si hay otro cercano, mediante una orden process:

.central.s process “Utilities Variables CreateAlwaysNewPoint 1”

Para crear la curva, a partir de una serie de puntos, se usa la instrucción

.central.s process “Geometry Create NurbsLine”

seguida por las coordenadas de los puntos

.central.s process “$x $y $z”

Al terminar la creación de la curva se restaura el valor inicial de la variable CreateAlwaysNewPoint.

Para obtener las coordenadas se hace un bucle for, de forma que t varíe t0 a t1, con el número de divisiones deseado (número de puntos-1). Los valores de t, x, y, z se obtienen mediante la instrucción expr mencionada anteriormente.

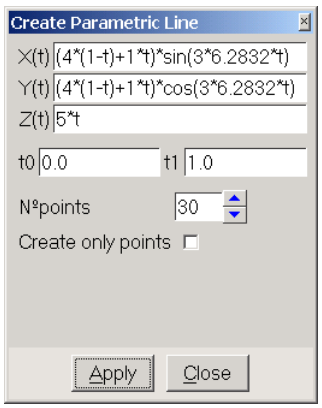
Finalmente se ha añadido al programa básico algunas instrucciones adicionales, para inicializar el valor de las variables, mejorar la estética de la ventana, evitar redibujados al crear cada punto, así como para controlar posibles errores del usuario al entrar la fórmulas.

Para ayudar al usuario, se ha incluido una pequeño texto de ayuda, acerca de la sintaxis válida en las fórmulas, mediante la instrucción

GidHelp “nombre de widget” “Texto de ayuda”

Este texto aparecerá si el usuario pica el “widget” pulsando con el botón derecho del ratón.

El código final de la aplicación es el siguiente:




```
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
    #creación de la ventana principal
    InitWindow $w "Create Parametric Line" ParametricWindowGeom
    #creación de los witgets
    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
    #empaquetamiento de los 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"
}
```

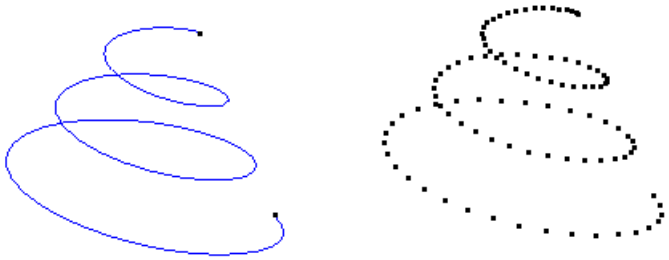
```
proc IncrementNPT { } {
    global ParametricPriv
    set ParametricPriv(NumberPointsT) [expr $ParametricPriv(NumberPointsT)+1]
}

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)
   #validación de datos
    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
    }
    # sustitución de t por $t en las fórmulas
    foreach {f} {fx fy fz} {
            regsub -all -nocase {\mt\M} [set $f] {$t} $f
    }
    #desactivar el redibujado y poner el cursor del ratón en símbolo de esperar
    $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
    }
```

```
    #reactivar el dibujado, poner el cursor en modo normal  y redibujar
    .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. Ejemplos de uso

Inicialmente, como muestra, se rellenan las fórmulas con la expresión de una hélice cónica. Esta hélice comienza con un radio R_0=4 y termina con un radio R_1=1, dando N=3 vueltas al moverse desde t=0.0 hasta t=1.0, la altura se hace variar desde 0 hasta H=5.

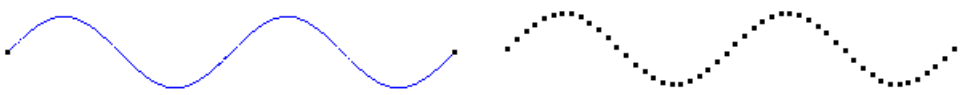
$$\begin{cases} x(t) = (R_0 \cdot (1-t) + R_1) \cdot \sin(N \cdot 2\pi \cdot t) \\ y(t) = (R_0 \cdot (1-t) + R_1) \cdot \cos(N \cdot 2\pi \cdot t) \\ z(t) = H \cdot t \end{cases}$$



Ejemplo de hélice cónica creando una curva o solamente los puntos

A continuación, se muestra otro ejemplo de uso, con una gráfica bidimensional y=f(x), usando la función seno, con el parámetro t variando de 0.0 a 4.0*pi

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



Ejemplo de función sinusoidal

6. Información adicional sobre Tcl/Tk

Existen multitud de páginas web que ofrecen información, tutoriales, etc. acerca del lenguaje de programación Tcl, la más destacadas es:

http://tcl.activestate.com

En la dirección anteriores, hay documentación y tutoriales en inglés, puede encontrarse un tutorial interesante de iniciación a Tcl/Tk en castellano en esta otra dirección (formato html):

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

Para obtener información acerca de GiD (manuales, tutoriales, etc), puede consultarse esta dirección:

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