Extensión de GiD mediante Tcl/Tk

Herramienta de edición de malla

Este tutorial consistirá en agregar a GiD una herramienta que algunos usuarios pueden hacer echado en falta en alguna ocasión, mediante la cual se pueda crear o borrar manualmente nodos o elementos de una malla.

Esta herramienta no será intrínseca de GiD, no está incluida en el ejecutable compilado (cuyo núcleo está programado en C++), sino que tratará de aparentarlo, aunque en realidad la programaremos mediante el lenguaje “script” Tcl/Tk.

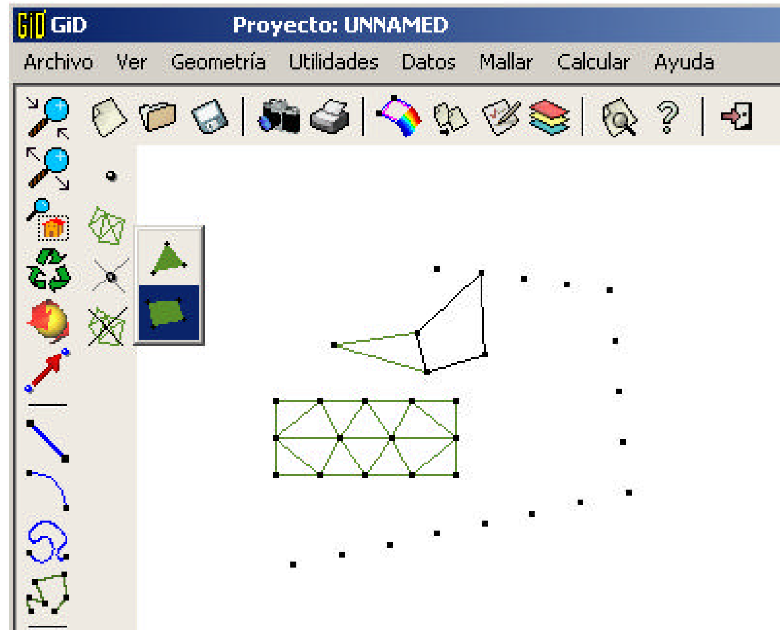


TABLA DE CONTENIDOS

1. DESCRIPCIÓN DE LA NUEVA HERRAMIENTA 92

2. USO DE TCL DESDE UN 'TIPO DE PROBLEMA' 93

3. CREACIÓN BÁSICA DE NODOS Y ELEMENTOS 94

4. MEJORAS DEL PROCEDIMIENTO BÁSICO 97

5. DIBUJADO DIRECTO MEDIANTE OPEN GL 99

6. CREACIÓN DE UNA BARRA DE HERRAMIENTAS 101

7. ORIGEN DE LOS PROCEDIMIENTOS USADOS 102

8. LISTADO DEL CÓDIGO DEFINITIVO 103

1. DESCRIPCIÓN DE LA NUEVA HERRAMIENTA

En este apartado crearemos una nueva utilidad para crear malla de forma manual. Esta forma manual de construcción de malla puede ser útil en determinados casos sencillos, especialmente trabajando en dos dimensiones.

Permitiremos crear nuevos nodos o destruir los que se seleccionen, así como construir elementos pulsando los nodos que los forman.

La intención de este ejemplo es mostrar al desarrollador de GiD la forma de realizar varias operaciones muy comunes:

- Uso de Tcl desde un ‘tipo de problema’ (eventos InitGiDProject, etc)
- Creación y modificación de los menús
- Selección de entidades de GiD o coordenadas de pantalla
- Instrucciones Tcl especiales agregadas a GiD para crear malla, etc.
- Mostrar frases en la barra de mensajes inferior, aparentar estado de espera, mostrar una ventana de bienvenida, etc.
- Comprobar que la versión de GiD es apropiada para soportar el código
- Dibujado directo en la ventana central mediante instrucciones de Open GL
- Creación de una barra de iconos integrada en GiD.

2. USO DE TCL DESDE UN 'TIPO DE PROBLEMA'

La primera cuestión que se le presenta a un programador que quiere extender GiD mediante el potente lenguaje Tcl/Tk es: ¿Dónde empiezo a escribir, y cómo se ejecutará mi código?

Un 'tipo de problema' de GiD, no es más que un directorio "<nombre>.gid", dentro del cual hay una serie de ficheros ASCII que definen como se personaliza el programa. Por ejemplo, llamaremos a nuestro tipo de problema de este ejemplo "CreateMesh.gid". Uno de sus posibles ficheros es "CreateMesh.tcl", y el punto de entrada de nuestro código Tcl es la función InitGIDProject, que tiene el siguiente prototipo:

proc InitGIDProject { dir }

Esta función es un "evento" que genera GiD cuando el usuario carga el tipo de problema, y dir contiene la ruta completa a su directorio.

Por tanto, nuestro código Tcl debe implementar dicho procedimiento, aprovechando para modificar los menús, etc. En el primer listado añadiremos varias opciones nuevas al menú "Mesh"

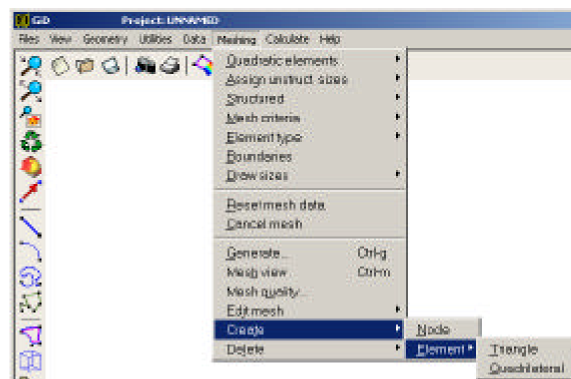
```
proc InitGIDProject { dir } {
    InsertMenuOption "Meshing" "Create" 15 {} PRE
    InsertMenuOption "Meshing" "Create>Node" 0 CreateNode PRE
    InsertMenuOption "Meshing" "Create>Element" 1 {} PRE
    InsertMenuOption "Meshing" "Create>Element>Triangle" 0 { CreateElement triangle } PRE
    InsertMenuOption "Meshing" "Create>Element>Quadrilateral" 1 \
           { CreateElement quadrilateral } PRE
    InsertMenuOption "Meshing" "Delete" 16 {} PRE
    InsertMenuOption "Meshing" "Delete>Node" 0 DeleteNode PRE
    InsertMenuOption "Meshing" "Delete>Element" 1 DeleteElement PRE
    UpdateMenus
}

proc EndGIDProject {} {
}


proc CreateElement { type } {
    WarnWin "CreateElement $type"
}
```

Se puede agregar una opción a un menú mediante el comando InsertMenuOption, que tiene como parámetros la etiqueta del menú a modificar, la etiqueta del nuevo submenú, el índice numérico donde se hace la inserción, el procedimiento a llamar al ser pulsado, y si debe mostrarse en preproceso, en postproceso o en ambos.

Obsérvese que los submenús se separan con el símbolo ">", por ejemplo: "Create>Element>Triangle", por tanto esto implica como restricción particular de GiD que una etiqueta de menú no puede contener dicho símbolo.

Al terminar la inserción de menús se llama a UpdateMenus para hacerlo efectivo.

Al cargar este tipo de problema (data->Problemtype->CreateMesh), puede verse el menú modificado con las nuevas opciones



Por el momento sólo hemos definido el procedimiento CreateElement, que únicamente muestra una ventana de mensajes modal, llamando al procedimiento WarnWin (particular de GiD, no es estándar de tcl), si se pulsa otra de las nuevas opciones se producirá un error tcl de llamada a procedimiento no definido.

EndGIDProject, es otro evento lanzado por GiD para permitir al "tipo de problema" responder de forma apropiada (restaurar los cambios, etc).

3. CREACIÓN BÁSICA DE NODOS Y ELEMENTOS

Ahora vamos a añadir la funcionalidad básica para crear nodos y elementos.

Para crear nodos se necesita pedir al usuario que especifique unas coordenadas, para ello puede usarse la función tcl GidUtils::GetCoordinates, definida dentro del fichero scripts\dev_kit.tcl (este fichero contiene funciones interesantes para quien desee ampliar GiD desde tcl)

Una vez obtenida la coordenada, debe crearse un nodo, para ello existe una función tcl: GiD_Mesh create node append $coord.

Análogamente para la creación de un elemento se necesita la selección de nodos existentes. Para seleccionar entidades existentes, puede usarse el procedimiento GidUtils::PickEntities nodes single

Puede insertarse una frase en la barra de mensajes inferior mediante GidUtils::SetWarnLine $texto

Este sería el listado de una versión simple para crear o borrar nodos y elementos:

```
proc InitGIDProject { dir } {
    InsertMenuOption [_ "Meshing"] [= "Create"] 15 {} PRE
    InsertMenuOption [_ "Meshing"] [= "Create"]>[= "Node"] 0 CreateNode PRE
    InsertMenuOption [_ "Meshing"] [= "Create"]>[= "Element"] 1 {} PRE
    InsertMenuOption [_ "Meshing"] [= "Create"]>[= "Element"]>[= "Triangle"] 0 \
           { CreateElement triangle } PRE
    InsertMenuOption [_ "Meshing"] [= "Create"]>[= "Element"]>[= "Quadrilateral"] 1 \
           { CreateElement quadrilateral } PRE
    UpdateMenus
}

proc EndGIDProject {} {
}

proc CreateNode { } {
    set coord [GidUtils::GetCoordinates [= "Enter node coordinates"] NoJoin]
    if { $coord != "" } {
       GiD_Mesh create node append $coord
       .central.s process redraw
    } else {
       GidUtils::SetWarnLine [= "Leaving create nodes"]
    }
}

proc CreateElement { type } {
    if { $type == "triangle" } {
       set nnode 3
    } elseif { $type == "quadrilateral" } {
       set nnode 4
    } else {
       GidUtils::SetWarnLine [= "Unexpected element type '%s'" $type]
       return
    }
    for {set i 0 } { $i < $nnode } { incr i } {
       GidUtils::SetWarnLine [= "Enter node %s" [expr $i+1]]
       #set olddisablewarnline [GidUtils::DisableWarnLine]
       set ni [GidUtils::PickEntities nodes single]
       #if { !$olddisablewarnline } { GidUtils::EnableWarnLine }
```

```
        if { $ni == "" } {
            GidUtils::SetWarnLine [= "Leaving create element"]
            return
        }
        lappend nodes $ni
    }
    GiD_Mesh create element append $type $nnode $nodes
    .central.s process redraw
}

proc DeleteNode { } {
    set nodeslist [GidUtils::PickEntities nodes multiple]
    if { $nodeslist != "" } {
        foreach i [GidUtils::UnCompactNumberList $nodeslist] {
            GiD_Mesh delete node $i
        }
        .central.s process redraw
    } else {
        GidUtils::SetWarnLine [= "Leaving delete nodes"]
    }
}

proc DeleteElement { } {
    set elemslist [GidUtils::PickEntities elements multiple]
    if { $elemslist != "" } {
        foreach i [GidUtils::UnCompactNumberList $elemslist] {
            GiD_Mesh delete element $i
        }
        .central.s process redraw
    } else {
        GidUtils::SetWarnLine [= "Leaving delete elements"]
    }
}
```

Nótese un detalle, ahora las líneas relacionadas con la creación de menús han cambiado ligeramente:

Por ejemplo, en vez de: InsertMenuOption "Meshi ng" "Create" 15 {} PRE

Ahora pone: InsertMenuOption [_ "Meshing"] [= "Create"] 15 {} PRE

Este detalle está relacionado con la reciente posibilidad de GiD de ser multilingüe. Los textos mostrados en menús, etc. dependerán del lenguaje seleccionado. Por tanto, si el lenguaje actual es castellano en vez de inglés, la función InsertMenuOption fallaría porque no encontrará el menú “Meshing”, que ahora aparecerá como “Mallar”. El procedimiento “subrayado” busca en tiempo de ejecución el texto correspondiente a la frase en lengua inglesa (se adopta el inglés como texto del código fuente), y al encerrar una función tcl entre corchetes [ ] se evalúa.

Además hay otro detalle de traducción a tener en cuenta, hay dos bases de datos de traducción: la de los textos propios de GiD, cuya traducción se invoca con [_], y otra particular del "tipo de problema", que debe invocarse análogamente con el procedimiento "igual" [=]. La palabra "Meshing" debe traducirse con _ porque es un menú estándar de GiD, en cambio "Create" es una opción de submenú agregada por el "tipo de problema", y que debe traducirse con = . El creador del "tipo de problema" debe incorporar una colección de bases de datos de traducciones (por ejemplo en.msg, es.msg, etc.) en el directorio \msgs del propio "tipo de problema" (GiD cargará estas traducciones al leer dicho tipo de problema)

4. MEJORAS DEL PROCEDIMIENTO BÁSICO

Mediante el código anterior ya es posible crear y eliminar malla, pero aún quedan algunos detalles por pulir:

1. Por defecto en GiD no se dibujan los nodos, porque usualmente hay muchos, y se vería la imagen casi negra. En cambio en este caso, cuando el usuario crea un nodo, es fundamental que lo vea para tener un eco de su acción.

Por ello se activará la variable de GiD ShowFigures(Nodes)

```
set prevdrawnodes [.central.s info variables ShowFigures(Nodes)]
if { $prevdrawnodes == "0" } {
    .central.s process utilities variables ShowFigures(Nodes) 1 escape
    .central.s process redraw
}
```

Se almacena el valor previo de la variable, se establece el nuevo valor temporalmente, y se restaurar al salir del procedimiento de creación de nodos:

```
if { $prevdrawnodes != [.central.s info variables ShowFigures(Nodes)] } {
     .central.s process utilities variables ShowFigures(Nodes) $prevdrawnodes escape
     .central.s process redraw
}
```

2. Si se quiere hacer una serie de nodos, es muy incómodo tener que invocar la función de creación cada vez. Es más operativo que la función se repita automáticamente al terminar, terminando un escape sin seleccionar nada.

Para ello, se añadirá en el punto de salida normal de la función una llamada a si misma, pero evitando dicha autoinvocación cuando el usuario desea terminar, ya que de otro modo se entraría en un bucle infinito:

Análogamente pasa con el caso de los elementos.

3. Si en GiD está activada la opción de elementos cuadráticos, en vez de pedir tres nodos para definir un elemento, hay que pedir seis (en orden jerárquico: primero las tres esquinas y luego los puntos intermedios)

```
set quadratic [lindex [.central.s info project] 5]
if { $type == "onlypoints" } {
    set nnode 1
} elseif { $type == "linear" } {
    if { $quadratic == "0" } {
        set nnode 2
    } else {
        set nnode 3
    }
} elseif { $type == "triangle" } {
    if { $quadratic == "0" } {
        set nnode 3
    } else {
        set nnode 6
    }
} elseif { $type == "quadrilateral" } {
    if { $quadratic == "0" } {
        set nnode 4
    } elseif { $quadratic == "1" } {
        set nnode 8
    } else {
        set nnode 9
    }
} else {
    #can also accept: tetrahedra hexahedra prism
    GidUtils::SetWarnLine [= "Bad element type '%s'" $type]
    return
}
```

4. El comando GiD_Mesh, así como el resto de comandos tcl no quedan registrados en ninguna parte, por tanto no funcionará correctamente la utilidad “Undo” para deshacer cambios.

Nota: sólo se registran los comandos .central.s process “comando”

Podemos forzar desde tcl a registrar y evaluar un comando tcl mediante GidUtils::EvalAndEnterInBatch <comando tcl>

Por ejemplo, en vez de

GiD_Mesh create node append $coord

Usaremos

GidUtils::EvalAndEnterInBatch GiD_Mesh create node append $coord.

5. DIBUJADO DIRECTO MEDIANTE OPEN GL

En las últimas versiones de GiD, se ha añadido un nuevo comando tcl: drawopengl, que tiene grandes posibilidades para extender GiD. Ahora se pueden utilizar prácticamente todas las instrucciones de la librería gráfica Open GL desde un "script" tcl.

Aprovecharemos esta nueva instrucción para mejorar el procedimiento de creación de un elemento. Hasta este momento, a medida que el usuario entra los nodos de un elemento, le aparece en la barra inferior un mensaje: "Enter node 1", "Enter node 2", etc. Indicándole que nodo va a entrar. Sería mucho más intuitivo si a medida que va entrando nodos se va dibujando en la medida de lo posible el elemento.

Crearemos un procedimiento, llamado DrawTemporaryElem, donde se unirán con líneas las coordenadas disponibles del elemento.

```
proc DrawTemporaryElem { } {
    global CreateMeshPriv
    if { ![info exists CreateMeshPriv(inode)] || $CreateMeshPriv(inode) == "-1" } return

    set color "0 0 0"
    drawopengl draw -color $color -begin lineloop
        for { set i 0 } { $i <= $CreateMeshPriv(inode) } { incr i } {
            drawopengl draw -vertex $CreateMeshPriv(coord,$i)
    }
    drawopengl draw -end
}
```

Las coordenadas de los nodos se guardarán en una variable global, ya que se rellenan en el procedimiento CreateElement, y se usan en otro procedimiento. Conviene ser cuidadoso con la nomenclatura usada en las variables globales, ya que puede haber conflictos con otros procedimientos que usen el mismo nombre.

Por ejemplo, para almacenar cual es el número local de nodo que se debe entrar, sería muy mala elección guardarlo en una variable global llamada “inode”. En nuestro caso es una buena práctica guardar todas las variables globales relativas a nuestro tipo de problema en un mismo “array”, cuyo nombre ya de idea de su contenido. Por ejemplo usaremos un “array” global, llamado CreateMeshPriv, y en concreto el número local de nodo será CreateMeshPriv(inode), y las coordenadas de los nodos entrados serán CreateMeshPriv(coord,0) CreateMeshPriv(coord,1) ,etc.

Con este estilo de denominación de variables disminuirá mucho la posibilidad de conflictos (podrían emplearse otros esquemas, como usar “namespaces”)

La función de dibujado es muy simple, usa la instrucción drawopengl draw , que acepta como parámetros casi cualquier orden estándar de OpenGL, eliminando de su notación el prefijo “gl” ó “GL_” y otros pequeños cambios, por ejemplo en este caso se han usado los siguientes comandos:

Estilo estándar OpenGL C/C++	Estilo simplificado drawopengl draw
glColor3i(r,g,b)	-color $color
glBegin(GL_LINE_LOOP)	-begin lineloop
glVertex3d(x,y,z)	-vertex
glEnd()	-end

Se han incluido en la instrucción drawopengl algunos parámetros especiales para GiD, por ejemplo:

drawopengl register <nombre_procedimiento_tcl>

drawopengl unregister $id

Sirven para registrar un procedimiento tcl para que sea llamado automáticamente cuando GiD redibuja la pantalla. Este procedimiento devuelve un identificador que sirve para eliminar el procedimiento del registro mediante drawopengl unregister

El procedimiento de dibujo del elemento es muy rudimentario, en un caso real convendría dibujar un elemento dinámico, con un nodo moviéndose a la posición del cursor. Pintando en “foreground” en vez de en “background” y redibujando, etc.

6. CREACIÓN DE UNA BARRA DE HERRAMIENTAS

Otro detalle interesante para un desarrollador de un “tipo de problema” es poder añadir barras de herramientas especiales, para facilitar el acceso a sus funciones.

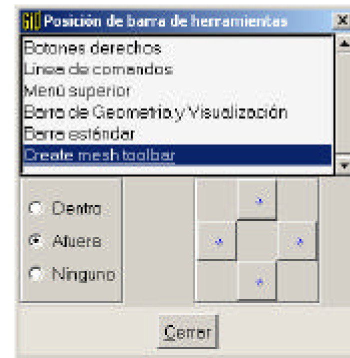
Por ejemplo añadiremos una barra de herramientas de edición de malla, con iconos para crear y borrar nodos y elementos.

Construiremos nuestra barra en el procedimiento CreateMeshToolbar, donde rellenaremos varias listas conteniendo los nombres de ficheros de las imágenes, los comandos a invocar, y el texto de ayuda emergente asociado. Con estos y otros datos (nombre de la barra, título, posición, etc.), se llama al procedimiento CreateOtherBitmaps, al cual se le pasa también el nombre del procedimiento de reconstrucción: CreateMeshToolbar.

El nombre interno de la barra, según un convenio adoptado, debe terminar en “bar”, por ejemplo en este caso se usa:

```
set CreateMeshPriv(toolbarname) createmeshbar
```

La barra debe registrarse para poder ser manejada desde la ventana que aparece con Utilities->Graphical->Toolbars. Esto se hace mediante AddNewToolbar.



A la derecha puede verse dicha ventana de control de la posición de las barras de herramientas.

Se aprovechará el evento InitGIDProject para agregar nuestra barra, y el evento EndGIDProject para eliminar dicha barra, así como las variables y procedimientos particulares del tipo de problema. Debe usarse el procedimiento ReleaseToolbar para que se deje de considerar en la ventana de las barras de herramientas.

7. ORIGEN DE LOS PROCEDIMIENTOS USADOS

Al principio puede resultar algo confuso, para el desarrollador que examina un código ajeno, saber que procedimientos son puramente lenguaje Tcl/Tk estándar y cuales no. Este apartado trata de aclarar un poco este hecho.

Procedimientos definidos internamente en C++ dentro de GiD:

GiD_Mesh
drawopengl

Eventos GiD-tcl lanzados internamente desde C++

InitGIDProject
EndGIDProject

Procedimientos definidos externamente en ficheros tcl de GiD:

InsertMenuOption
UpdateMenus
_
=
GidUtils::GetCoordinates
GidUtils::PickEntities

GidUtils::UnCompactNumberList

GidUtils::EvalAndEnterInBatch

GidUtils::SetWarnLine

GidUtils::DisableWarnLine

GidUtils::EnableWarnLine

CreateOtherBitmaps

AddNewToolbar

ReleaseToolbar

Opciones del “widget” .central.s (widget Tk especial definido en el núcleo de GiD)

.central.s info

.central.s process

El resto de procedimientos son Tcl estándar, y su descripción puede encontrarse en la ayuda de Tcl/Tk (incluida en la herramienta RamDebugger de CIMNE)

8. LISTADO DEL CÓDIGO DEFINITIVO

Este es el código tcl definitivo, aunque evidentemente podría seguir mejorándose:

```
proc InitGIDProject { dir } {
    global CreateMeshPriv
    InsertMenuOption [_ "Meshing"] [= "Create"] 15 {} PRE
    InsertMenuOption [_ "Meshing"] [= "Create"]>[= "Node"] 0 CreateNode PRE
    InsertMenuOption [_ "Meshing"] [= "Create"]>[= "Element"] 1 {} PRE
    InsertMenuOption [_ "Meshing"] [= "Create"]>[= "Element"]>[= "Triangle"] 0 \
            { CreateElement triangle } PRE
    InsertMenuOption [_ "Meshing"] [= "Create"]>[= "Element"]>[= "Quadrilateral"] 1 \
            { CreateElement quadrilateral } PRE
    InsertMenuOption [_ "Meshing"] [= "Delete"] 16 {} PRE
    InsertMenuOption [_ "Meshing"] [= "Delete"]>[= "Node"] 0 DeleteNode PRE
    InsertMenuOption [_ "Meshing"] [= "Delete"]>[= "Element"] 1 DeleteElement PRE
    UpdateMenus
    CreateMeshToolbar $dir
}

proc EndGIDProject {} {
    global CreateMeshPriv
    if { [info exists CreateMeshPriv(drawopengl)] } {
       catch { drawopengl unregister $CreateMeshPriv(drawopengl) }
    }
    DeleteMeshToolbar

    #delete problemtype defined procedures
    foreach i {CreateNode DrawTemporaryElem CreateElement DeleteNode DeleteElement \
            CreateMeshToolbar} {
       rename $i ""
    }
    catch { unset CreateMeshPriv }
}

proc CreateMeshToolbar { dir { type "DEFAULT INSIDELEFT"} } {
    global CreateMeshBitmapsNames CreateMeshBitmapsCommands CreateMeshBitmapsHelp
    global CreateMeshPriv
```

```
    set CreateMeshBitmapsNames(0) [list \
            "createnode.gif" \
            "createelement.gif" \
            "deletenode.gif" \
            "deleteelement.gif"]
    set CreateMeshBitmapsCommands(0) [list \
            "-np- CreateNode" \
            "" \
            "-np- DeleteNode"\
            "-np- DeleteElement"]
    set CreateMeshBitmapsHelp(0) [list \
            [= "Creates a new node"] \
            [= "Creates a new element"] \
            [= "Delete the selected nodes"] \
            [= "Delete the selected elements"]]
    set CreateMeshBitmapsNames(0,1) [list \
            "createtriangle.gif" \
            "createquadrilateral.gif" ]
    set CreateMeshBitmapsCommands(0,1) [list \
            "-np- CreateElement triangle" \
            "-np- CreateElement quadrilateral"]
    set CreateMeshBitmapsHelp(0,1) [list \
            [= "Creates a new triangle element"] \
            [= "Creates a new quadrilateral element"]]
    #CreateOtherBitmaps { Name title BitMapsNames BitMapsCommands BitmapsHelp BitmapsPath \
    #StartFunc { what "DEFAULT" } { prefix PrePost} }
    # valid prefix values: Pre, Post, PrePost
    set CreateMeshPriv(toolbarname) createmeshbar
    set CreateMeshPriv(toolbartitle) [= "Create mesh toolbar"]
    set prefix Pre
    set CreateMeshPriv(toolbarwin) [CreateOtherBitmaps $CreateMeshPriv(toolbarname) \
            $CreateMeshPriv(toolbartitle) CreateMeshBitmapsNames CreateMeshBitmapsCommands \
            CreateMeshBitmapsHelp [file join $dir bitmaps] [list CreateMeshToolbar $dir] \
            $type $prefix]
    AddNewToolbar $CreateMeshPriv(toolbarname) \
            "${prefix}${CreateMeshPriv(toolbarname)}WindowGeom" \
            [list CreateMeshToolbar $dir] $CreateMeshPriv(toolbartitle)
}

proc DeleteMeshToolbar {} {
    global CreateMeshPriv

    ReleaseToolbar $CreateMeshPriv(toolbarname)
    if { [winfo exists $CreateMeshPriv(toolbarwin)] } {
        destroy $CreateMeshPriv(toolbarwin)
    }
    array unset CreateMeshPriv
}

proc CreateNode { } {
    #if the nodes are not drawn is changed temporary this setting
    set prevdrawnodes [.central.s info variables ShowFigures(Nodes)]
    if { $prevdrawnodes == "0" } {
        .central.s process utilities variables ShowFigures(Nodes) 1 escape
        .central.s process redraw
    }
    set coord [GidUtils::GetCoordinates [= "Enter node coordinates"] NoJoin]
    if { $coord != "" } {
        GidUtils::EvalAndEnterInBatch GiD_Mesh create node append $coord
        #GidUtils::SetWarnLine [= "Node created"]
        .central.s process redraw
        CreateNode
    } else {
        GidUtils::SetWarnLine [= "Leaving create nodes"]
        if { $prevdrawnodes != [.central.s info variables ShowFigures(Nodes)] } {
            .central.s process utilities variables ShowFigures(Nodes) $prevdrawnodes
            .central.s process escape redraw
        }
    }
}
```

```
proc DeleteNode { } {
    #if the nodes are not drawn is changed temporary this setting
    set prevdrawnodes [.central.s info variables ShowFigures(Nodes)]
    if { $prevdrawnodes == "0" } {
        .central.s process utilities variables ShowFigures(Nodes) 1 escape
        .central.s process redraw
    }
    set nodeslist [GidUtils::PickEntities nodes multiple]
    if { $nodeslist != "" } {
        foreach i [GidUtils::UnCompactNumberList $nodeslist] {
            GidUtils::EvalAndEnterInBatch GiD_Mesh delete node $i
        }
        .central.s process redraw
        DeleteNode
    } else {
        GidUtils::SetWarnLine [= "Leaving delete nodes"]
        if { $prevdrawnodes != [.central.s info variables ShowFigures(Nodes)] } {
            .central.s process utilities variables ShowFigures(Nodes) $prevdrawnodes escape
            .central.s process redraw
        }
    }
}

proc DrawTemporaryElem { } {
    global CreateMeshPriv
    if { ![info exists CreateMeshPriv(inode)] || $CreateMeshPriv(inode) == "-1" } return

    set color "0 0 0"
    drawopengl draw -color $color -begin lineloop
        for { set i 0 } { $i <= $CreateMeshPriv(inode) } { incr i } {
            drawopengl draw -vertex $CreateMeshPriv(coord,$i)
    }
    drawopengl draw -end
}

proc DeleteElement { } {
    set elemslist [GidUtils::PickEntities elements multiple]
    if { $elemslist != "" } {
        foreach i [GidUtils::UnCompactNumberList $elemslist] {
            GidUtils::EvalAndEnterInBatch GiD_Mesh delete element $i
        }
        .central.s process redraw
        DeleteElement
    } else {
        GidUtils::SetWarnLine [= "Leaving delete elements"]
    }
}


proc CreateElement { type } {
    global CreateMeshPriv
    set quadratic [lindex [.central.s info project] 5]
    if { $type == "onlypoints" } {
        set nnode 1
    } elseif { $type == "linear" } {
        if { $quadratic == "0" } {
            set nnode 2
        } else {
            set nnode 3
        }
    } elseif { $type == "triangle" } {
        if { $quadratic == "0" } {
            set nnode 3
        } else {
            set nnode 6
        }
    } elseif { $type == "quadrilateral" } {
        if { $quadratic == "0" } {
            set nnode 4
        } elseif { $quadratic == "1" } {
```

```
            set nnode 8
        } else {
            set nnode 9
        }
    } else {
        #can also accept: tetrahedra hexahedra prism
        GidUtils::SetWarnLine [= "Bad element type '%s'" $type]
        return
    }
    #if the nodes are not drawn is changed temporary this setting
    set prevdrawnodes [.central.s info variables ShowFigures(Nodes)]
    if { $prevdrawnodes == "0" } {
        .central.s process utilities variables ShowFigures(Nodes) 1 escape
        .central.s process redraw
    }
    set CreateMeshPriv(nnode) $nnode
    set CreateMeshPriv(inode) -1
    set CreateMeshPriv(drawopengl) [drawopengl register DrawTemporaryElem]

    set nodes ""
    for {set i 0 } { $i < $nnode } { incr i } {
        GidUtils::SetWarnLine [= "Enter node %s" [expr $i+1]]
        set olddisablewarnline [GidUtils::DisableWarnLine]
        set ni [GidUtils::PickEntities nodes single]
        if { !$olddisablewarnline } { GidUtils::EnableWarnLine }
        if { $ni == "" } {
            GidUtils::SetWarnLine [= "Leaving create element"]
            if { $prevdrawnodes != [.central.s info variables ShowFigures(Nodes)] } {
                .central.s process utilities variables ShowFigures(Nodes) $prevdrawnodes
                .central.s process escape redraw
            }
            return
        }
        lappend nodes $ni
        set CreateMeshPriv(inode) $i
        set CreateMeshPriv(coord,$i) [lindex [.central.s info coordinates $ni mesh] 0]
        .central.s process redraw
    }
    catch { drawopengl unregister $CreateMeshPriv(drawopengl) }
    GidUtils::EvalAndEnterInBatch GiD_Mesh create element append $type $nnode $nodes
    #GidUtils::SetWarnLine [= "Element created"]
    .central.s process redraw
    CreateElement $type
}
```