A Tcl/Tk Add-on Script for Large Meshing Software

Wenny Wang
Pointwise, Inc., 213 S Jennings Avenue, Fort Worth, TX, 76104
wxh@pointwise.com

Abstract

Gridgen is meshing software used by engineers and scientists worldwide since 1984 to reliably generate high quality grids for engineering analysis. Gridgen Glyph, which is Gridgen's Tcl-based scripting language, provides full access to the functionalities available interactively in Gridgen. It allows users to extend Gridgen's functions as well as develop specialized meshing applications. This paper shows how a new Gridgen capability, automated block topology change, is developed through a script using Glyph, Tcl and Tk. This feature changes grid topologies in a matter of seconds while interactively changing grid topologies can require hours of a user's time.

Keywords: Glyph, connector, domain, block, O-H (butterfly) topology, structured mesh, topology propagating

1. Gridgen and Glyph Scripting

To conduct computational fluid dynamics (CFD) and finite element analysis (FEA), a two-dimensional (2D) or three-dimensional (3D) grid needs to be constructed for the geometry beforehand. Gridgen is a complete toolkit for generating grids with a variety of cell types (i.e., hexahedra, tetrahedra, prism and pyramid) that meet the simulation requirements.

During the construction of a grid in Gridgen, users will work with four types of hierarchical entities:

- *blocks*: volume grids
- *domains*: surface grids
- *connectors*: curve grids
- *database*: geometry data that defines the shape of the object being gridded

Typically the database is first obtained from a computer aided design (CAD) system and imported into Gridgen. It can also be created using Gridgen. However, the database is not required for Gridgen to generate grids. The remaining three grid entity types are created from the bottom of the hierarchy, connectors, up. This is usually known as the "bottom-up" meshing approach.

As an interactive code, Gridgen can grid practically anything, but at the price of user effort and time. As a response to user requests for streamlined gridding, Gridgen has been enhanced with Glyph, which is essentially Tcl with Gridgen-specific commands added to it. It contains approximately 230 default, display, grid construction, geometry, query, analysis software, and utility commands.

Glyph scripts are comprised of a series of commands. They may be functions native to Tcl/Tk, Gridgen-specific functions, or user defined functions. Being used to drive Gridgen in either interactive or batch mode, Glyph scripts allow users to develop specialized gridding applications with a customized user interface.

2. Problem Description

As shown in Fig. 1, an O-grid is a series of blocks with grid lines arranged into an "O" shape or a wrapping nature.

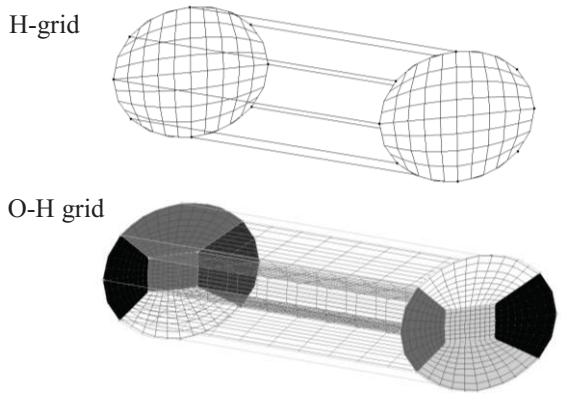


Fig. 1 H-grid and 5 block O-H grid

In contrast to the H-grid (Fig. 1), the O-H topology reduces skew (i.e., poor quality quads with large included angle) where a block corner must lie on continuous curves/surfaces (i.e., cylinder). It also improves efficiency of grid point clustering near walls. That is, it helps resolve the boundary layer locally near solid bodies without unnecessarily increasing overall grid point count.

Since Gridgen does not have automatic O-H grid creation capability, users have to modify a single H-block or H-blocks into the 5 block topology by hand: Center H-grid block and 4 O-grid blocks (Fig. 2).

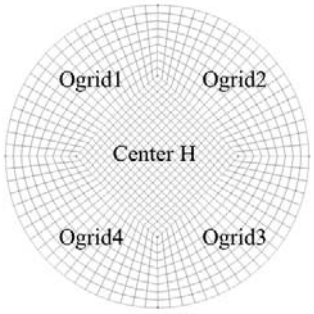


Fig. 2 O-H grid system

This can be very time consuming because new domains and connectors need to be created manually for constructing these blocks. Therefore, an add-on script was created for O-H grids to be constructed automatically.

3. Scripting Methodology

The add-on script, referred to as "Butterfly Maker", is written in Tcl/Tk 8.3 and Glyph. It has approximately 4,000 lines and 50 procedures.

For example, the following small procedure, *getConByNode*, is used for returning the connectors at a given node (Nodes are connector end points in Gridgen):

```
proc getConByNode { pt conList } {
  set tol [gg::tolNode]
  foreach con $conList {
    set pta [gg::conGetPt $con \
             -arc 0]
    set ptb [gg::conGetPt $con \
             -arc 1]

    set dist_a [GetDist $pt $pta]
    set dist_b [GetDist $pt $ptb]

    if { $dist_a < $tol } {
       return "$con $ptb"
    } elseif { $dist_b < $tol } {
       return "$con $pta"
    }
  }
}
```

This code segment loops through a given connector list, and singles out the connectors whose nodes are close enough to a given point. The commands that begin with "*gg::*" are the Glyph commands. For instance, "*gg::conGetPt*" returns the coordinates of a point on a connector where the arc length is 0. Standard Tcl/Tk interpreters are linked to Gridgen. This allows the Tcl/Tk commands to be understood by Gridgen when they are used in Glyph scripts.

3.1 Tk Widgets Design

The script uses a Tk interface to allow the user to select blocks to modify and enter some meshing parameters. Fig. 3 shows the user interface consisting of 4 frames:

A. Block selection
B. Propagating direction selection
C. Parameter input & preview
D. Execution

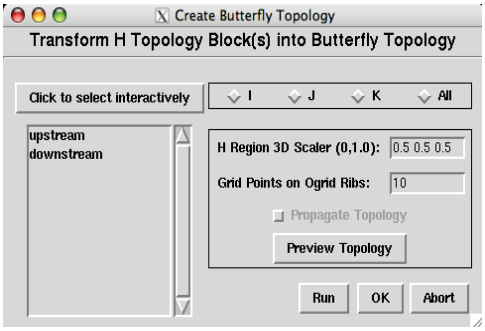


Fig. 3 Butterfly Maker user interface

Frame *A* enables block selection via either the list box or clicking on the button *Click to select interactively*. A scrollbar is added to the list box to display all of the blocks if there are many. The list box selection is bound to the <<ListboxSelect>> virtual event generated by the listbox widget. If the definition of this virtual event changes, all windows bound to it will respond immediately to the new definition.

```
bind .right.top.list \
     <<ListboxSelect>> { BlkSelect }
```

The *Click to select interactively* button is linked to the procedure *select* shown below:

```
proc select { } {
  global Mode Selected_Blocks
  global Propagate Direction
  global Structured_Blocks
  if { $Mode == "BLOCKS" && [llength
$Structured_Blocks] <= 0 } {
    tk_messageBox -title "Error: \
    No Blocks Available" -message\
    "There are no structured blocks \
    in the current grid." -type ok \
    -icon error
    return
  }

  CleanupCons
  wm withdraw .

  if { $Mode == "BLOCKS" } {
    set tSel [gg::dispPick BLOCK \
      -select [getDataList] \
      -explicit $Structured_Blocks \
      -message \
      "Please select blocks for
       butterfly topology to apply"]
   .right.top.list selection clear \
    0 end
    foreach i $tSel {
    .right.top.list  selection  set\
[lsearch   $Structured_Blocks   $i]
\[lsearch $Structured_Blocks $i]
    }
    if {[winfo exists .]} {
      wm deiconify .
    }
  }
}
```

The contents of the list box will be updated accordingly if any blocks are selected in the Gridgen display.

Once the target blocks are selected, the user specifies the propagating direction (*I/J/K/All*) in frame *B*. In Gridgen, the orientation of a structured block is represented in terms of (I, J, K). By selecting I, for example, the script will create 5 domains on the butterfly faces (i.e., *I_min* and *I_max*) of the selected blocks then propagate this topology in the I direction throughout all selected blocks.

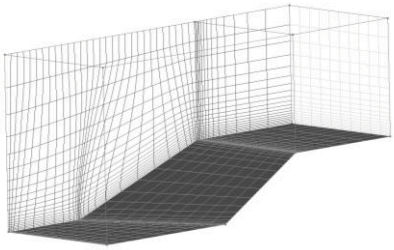
In frame *C*, the three dimensional scalar can be defined in the input field. This variable is a vector with three elements corresponding to I, J and K direction respectively. It allows the user to control the size of the O-grid portion. For grid refinement, users can specify the dimension of the connectors between the inner and outer boundaries of the O-grid in the *Grid Points on O-grid Ribs* input box.

The check button *Propagate Topology* is grayed out unless the propagating direction is specified. Toggling on this option will set the global variable *Propagate* to 1, align the IJK orientation of all selected blocks in the propagating direction, and call the procedure *redraw*. Pressing the *Preview Topology* button will update the display with the newly created internal connectors for topology review. For performance concerns, the new domains and blocks will not be created at this point.

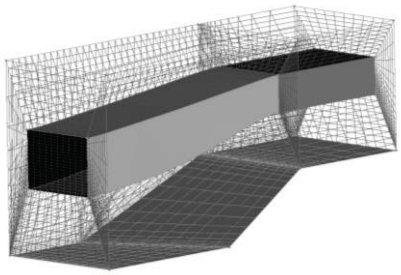
There are 3 buttons packed in frame *D*: *Run*, *OK* and *Abort*. The first two buttons are linked to similar actions except *OK* will perform the additional step of having Gridgen save the new grid and exit. *Abort* cleans up the temporary connectors then restores the original grid and settings, i.e., display and default attributes.

3.2 Approaches to Challenges

As shown in Fig. 4, to create O-H grids (b) based on two adjacent H-blocks (a), the following are the main steps:



a) Two adjacent H-topology blocks



b) New O-H block system

Fig. 4 Initial and target grids

1. Validate user input.
2. Obtain the block list in the I direction and realign blocks if necessary.
3. Determine which domains will be turned into "butterfly domains" and which will be kept.
4. Locate the center domain on each butterfly face and create its butterfly connectors.
5. Make sure no conflicts occur at the block interfaces.
6. Create new internal connectors in the I direction by which the new blocks are bounded.
7. Match up the distribution of these new connectors with their counterparts in the original blocks.
8. Assemble the butterfly domains and internal domains using the new connectors that were created.
9. Assemble the new blocks using the new domains and remaining original domains at the outer boundary.

3.2.1 User Input Diagnostics

Validating user input is especially important when multiple entries (i.e., value input and selection) are allowed in the user interface. With that in mind, the following items are checked before the main calculation begins:

1. Is the scaling factor valid? The three elements must be in the range of (0, 1).
2. Is the grid point number valid? The grid point number must be greater than 3.
3. Are there any temporary connectors that need to be eliminated? Temporary connectors are created for topology preview. They have to be removed whenever preview is updated.
4. Is the propagating block list valid? In order to have a valid propagating block list, all the selected blocks have to be

connected one to another in the propagating direction. The following procedure, *getParallelDirection,* is used to determine whether the two adjacent blocks (*blk1* and *blk2*) share a face in the propagating direction. Being used with several other procedures, it quickly sorts out the block/face orientation and execution will pause if invalid block/direction input is observed.

```
proc getParallelDirection { blk1
dirmm blk2 } {
  set face [gg::blkGetFace \
            $blk1 $dirmm]
  set retDir ""
  foreach dir {I J K} {
    if { [gg::blkGetFace \
          $blk2 ${dir}MIN]==$face|| \
         [gg::blkGetFace \
          $blk2 ${dir}MAX]==$face } {
       set retDir $dir
    }
  }

  if { $retDir == "" } {
     ErrorMsg "Faces of [gg::blkName
     $blk1] and [gg::blkName $blk2]
     in the propagating direction do
     not match. Not supported."
    quit
  } else { return $retDir }
}
```

If the user selects blocks before and after toggling on *Propagate Topology* in the user interface, duplicated blocks may be added to the propagating block list. To ensure no duplicates occur, the propagating block list needs to be updated dynamically. The following lines are added to the butterfly block creation procedure. This forces the procedure to immediately return if a duplicate is detected.

```
if { [lsearch -exact $createdBflyBlks
$blk] >= 0 } {
  puts "INFO: Duplicated blocks are \
       found in the propagating     \
       block list."
  return
}
```

The *createdBflyBlks* list is defined as a global variable that is updated whenever a butterfly block is created.

3.2.2 Universal Indexing

The indexing method plays an important role in creating new connectors, domains and blocks especially in complicated applications. For good performance, the indexing system should operate independently to the (I, J, K) system once it is defined.

```
if {$dir == "I"} {
  set max1 $jd
  set max2 $kd
  set ind3_max $id
  set ind1_min_face JMIN
  set ind1_max_face JMAX
  set ind2_min_face KMIN
  set ind2_max_face KMAX
  set ind3_min_face IMIN
  set ind3_max_face IMAX
} elseif {$dir == "J"} {
  set max1 $id
  set max2 $kd
  set ind3_max $jd
  set ind1_min_face IMIN
  set ind1_max_face IMAX
  set ind2_min_face KMIN
  set ind2_max_face KMAX
  set ind3_min_face JMIN
  set ind3_max_face JMAX
} else {
  set max1 $id
  set max2 $jd
  set ind3_max $kd
  set ind1_min_face IMIN
  set ind1_max_face IMAX
  set ind2_min_face JMIN
  set ind2_max_face JMAX
  set ind3_min_face KMIN
  set ind3_max_face KMAX
}
if { $dir == "ALL" } {
  set capMin 1
  set capMax 1
} else {
  set capMin 0
  set capMax 0
}
```

The string “ind3” stands for the propagating direction selected by the user. The strings “ind1” and “ind2” stand for the other two

directions respectively. Once defined, the indices for all the domains and connectors can be represented by two of the following variables:

- *ind1_min/max*
- *ind2_min/max*
- *ind3_min/max*
- *location on butterfly face (i.e., center, ogrid1, ogrid2, ogrid3 and ogrid4)*

Table 1 shows how the grid entities are related to each other through the universal indices.

Block Name	center
Domain Index	(center, ind3_min), (center, ind3_max), (center, ind1_min), (center, ind1_max), (center, ind2_min), (center, ind2_max)
Connector Index	(ind2_min, ind3_min), (ind1_max, ind3_min), (ind2_max, ind3_min), (ind1_min, ind3_min) .....
Block Name	ogrid1
Domain Index	(ogrid1, ind3_min), (ogrid1, ind3_max), (original face 1), (center, ind2_min), (corner1), (corner2)
Connector Index	(ind2_min, ind1_min), (ind1_min, ind3_max), (ind2_max, ind1_min), (ind1_min, ind3_min) .....

Tab. 1 Examples of grid entity indices

The following code segment assembles the "ogrid1" block using the universal indexing system:

```
gg::blkBegin -type STRUCTURED
  gg::faceBegin
    gg::faceAddDom  \
        $doms(ogrid1,ind3_min)
  gg::faceEnd
  gg::faceBegin
    gg::faceAddDom \
        $doms(ogrid1,ind3_max)
  gg::faceEnd
  gg::faceBegin
    foreach dm $face1 {
      gg::faceAddDom $dm
    }
  gg::faceEnd
  gg::faceBegin
    gg::faceAddDom  \
        $doms(center,ind2_min)
  gg::faceEnd
  gg::faceBegin
    gg::faceAddDom $doms(corner1)
  gg::faceEnd
  gg::faceBegin
    gg::faceAddDom $doms(corner2)
  gg::faceEnd
set blks(ogrid1) [gg::blkEnd]
```

This indexing method helps the script perform efficiently and makes the debugging process easier.

3.2.3 H Domain Locator

Locating the four corner points of the center H domain (Fig. 5) is one of the most important tasks in the script. Fig. 5 illustrates the location of the H domain on an *I_min* butterfly face. Given a scaling factor (*s1*, *s2*, *s3*), the four corners of the H domain are calculated and its orientation is kept consistent with the original H-block face.

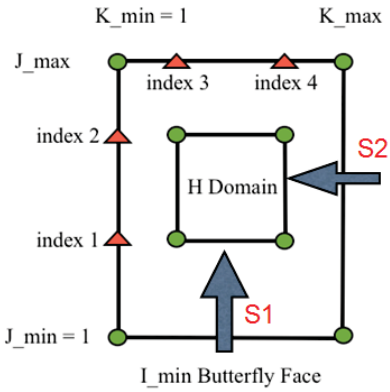


Fig. 5 H domain on butterfly face

The 1st and 2nd elements (s1 and s2) of the scaling factor are used for calculations in J and K direction respectively in this example. The 3rd element (s3), which is the I direction scaling factor, is applied only if the propagating direction is set to "All".

To determine the grid point indices of the two connectors in J and K directions (Fig. 5), a "point snapping" method is used for. The arc length between the *index 1* and *index 2*, for instance, is approximately the same as that of the connector in the J direction after being scaled.

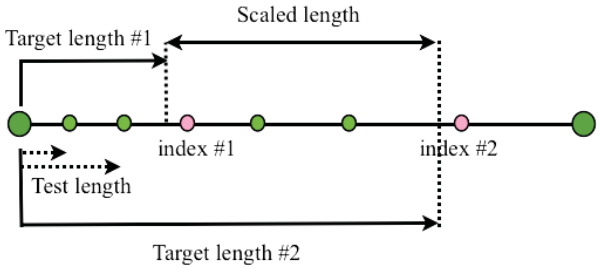


Fig. 6 Point snapping method

As show in Fig. 6, the arc length ("test length") between the beginning of the connector and each grid point is calculated. It is then compared with the two "target lengths" that determine where the two points (index 1 and 2) might be. A point is snapped when any of the following conditions is met:

- The test length is close to the target length #1.
- The test length is close to the target length #2.
- The difference of the test length and target length #1 is smaller than the local spacing.
- The difference of the test length and target length #2 is smaller than the local spacing.

The following code segment snaps points based on the above method:

```
set targetLength_1 [expr $iLength \
*(1.0-[lindex $oScaleFac 0]) / 2.0]

set targetLength_2 [expr $iLength \
- $targetLength_1 ]

set testLength 0.0
for { set ii 1 } { $ii < $max1 }  \
    {incr ii 1 } {
  set testLength [expr $testLength +
[lindex $iSpacing [expr $ii-1]]]
  if { [expr abs( $targetLength_1
       $testLength )] < $tol ||
      [expr abs( $targetLength_2
       $testLength )] < $tol ||
       expr abs( $testLength
       $targetLength_1)] < [lindex
       $iSpacing [expr $ii-1]] ||
      [expr abs( $testLength
       $targetLength_2)] < [lindex
       $iSpacing [expr $ii-1]] } {
    lappend corner1_Pts [expr $ii+1]
  }
}
```

A check is conducted once the points are added to the list. If no point is snapped, a backup scheme is enforced. With this arc length based method, the H domain is always placed in the center region regardless of the connector distributions.

If the propagating direction is set to "All", that requires the points on the 3rd direction (i.e., the I direction) to be snapped as well.

Once the indices are obtained, they are used for creating the connectors of the H domain. To ensure these connectors are constrained to the original face, they are created by a series of segments between adjacent grid points. Here is the code segment that achieves this:

```
gg::conBegin
  gg::segBegin -type 3D_LINE
    for {set ind1 $ind1_min} {$ind1
         <= $ind1_max} {incr ind1} {
      if [catch {gg::blkGetPt $blk \
         [blkGetIJK $dir $ind1 $ind2\
          $ind3]} pt] {
        gg::segEnd -nosave
        gg::conEnd -nosave
        DomPtFailed $blk $pt
        return 0
      }
      gg::segAddControlPt $pt
```

```
    }
  gg::segEnd
if [ catch {gg::conEnd} \
       con(ind2_min,ind3_min) ] {
  set con(ind2_min,ind3_min)     \
      [getConnectorByEndPoints   \
      [gg::blkGetPt $blk \
      [blkGetIJK $dir $ind1_min \
      $ind2 $ind3]] $pt]
}
```

Moreover, the butterfly domains are later projected onto the original surface for the shape information to be maintained.

3.2.4 Shared Edge Manager

This function is written to tackle complicated topological situations, for example, block faces consisting of multiple domains. This requires those connectors shared by adjacent faces (i.e., A and B in Fig. 7) to be reported and placed in order.

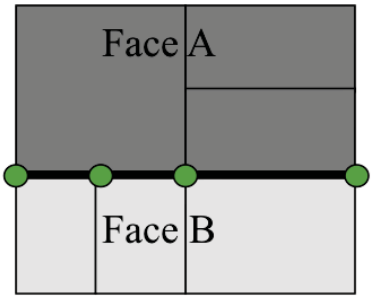


Fig. 7 Detect connectors on sharing edge

A quick way to do this is to rule out unqualified connectors (i.e., the internal connectors within a face) as each domain is inspected.

```
set dom_1 [lindex [gg::blkGetFace  \
          $blk $face1] 0]
set dom_2 [lindex [gg::blkGetFace \
          $blk $face2] 0]
set face_2_BoundCons {}
foreach dom $dom_2 {
  set edgeList [gg::domGetEdge $dom]
  foreach edge $edgeList {
    foreach con $edge {
      if { [lsearch $face_2_BoundCons
                $con] >= 0 } {
        # DO NOTHING.
      } else {
        lappend face_2_BoundCons $con
      }
    }
  }
}

set sharingCons {}
foreach dom $dom_1 {
  set edgeList [gg::domGetEdge $dom]
  foreach edge $edgeList {
    foreach con $edge {
      if { [lsearch $face_2_BoundCons
            $con] >= 0 } {
        lappend sharingCons $con
      }
    }
  }
}
```

For these shared connectors to be assembled into a domain edge, they must be connected in the order required by the Glyph commands. In other words, they need to be sorted if they are not arranged that way.

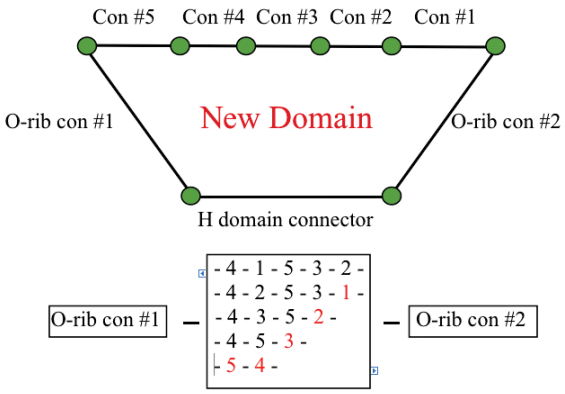


Fig. 8 Sort shared connectors

Fig. 8 demonstrates how the connector list is sorted within a few iterations to achieve the desired order: #5, #4, #3, #2 and #1. Given the initial arrangement: #4, #1, #5, #3, #2, for example, two connectors switch their locations at each iteration. One of them must be adjacent to the end of the sorted connector list.

```
proc edgeConsOrganizer {con1 Hcon
                        con2 edge} {
  set nodeTol [gg::tolNode]
  set H_pta [gg::conGetPt $Hcon \
             -arc 0.0]
  set H_ptb [gg::conGetPt $Hcon \
             -arc 1.0]
  set cor1_pta [gg::conGetPt $con1 \
                -arc 0.0]
  set cor1_ptb [gg::conGetPt $con1 \
                -arc 1.0]
  foreach Hnode [list $H_pta $H_ptb]{
    foreach node [list $cor1_pta \
                       $cor1_ptb] {
      if { [GetDist $node $Hnode] \
           > $nodeTol } {
        set edgeNode_1 $node
      }
    }
  }
  set conNum [llength $edge]
  set beginNode $edgeNode_1
  for { set i 0 } { $i < $conNum } {
incr i 1 } {
    set actualCon [lindex $edge $i]
    set temp [ getConByNode
$beginNode [lrange $edge $i end] ]
    set rightCon [lindex $temp 0]
    set rightConId [lsearch $edge
$rightCon]
    set beginNode [lrange $temp 1
end]
    if { [string equal $actualCon
$rightCon] != 1 } {
      set edge [lreplace $edge $i $i
$rightCon]
      set edge [lreplace $edge
$rightConId $rightConId $actualCon]
    }
  }
  return $edge
}
```

4. Future Work

With the release of the new meshing software, Pointwise, this script needs to be rewritten using the 2nd version of Glyph that uses an object-oriented paradigm. The following improvements are being considered:

- Handle other grid topologies: L-H, C-H and O grids around body.
- Add custom libraries for specific distributions of the O-rib connectors to be specified. For instance, applying a combined function of geometric and hyperbolic tangent distributions.
- Allow arbitrary H-domain locations (non-center) to be defined by users. The current script tends to place the H-domain at the center of a butterfly face.
- Optimize frequently used components to boost the script performance.
- Allow user to examine grid quality before saving.
- Improve the user interface using the advanced Tk widgets.
- Explore the possibility of parallelizing the script for large applications.

5. References

[1] J. Steinbrenner, C. Fouts and N. Wyman, 2002, Scripting Language as a Means of Automation in Gridgen, 8th International Grid Conference, Honolulu, HI.

[2] Pointwise. Inc, Gridgen Glyph Reference Manual.

[3] B. Welch, K. Jones and J. Hobbs, 2003, Practical Programming in Tcl and Tk, Pearson Education Inc.

6. Acknowledgement

I would like to express my thanks to those who helped me with the project and writing the paper. First and foremost, I'm grateful for Dr. Richard Matus's guidance, patience and support throughout the entire project. I would also like to thank Mr. Michael Jefferies and Ms. Carolyn Dear for taking time to review the paper. A special thank to Mr. Michael Jefferies for helping me better understand the Glyph environment.