A Tcl/Tk Add-on Script for Gridgen:

Butterfly Maker

– Wenny Wang, Pointwise, Inc.

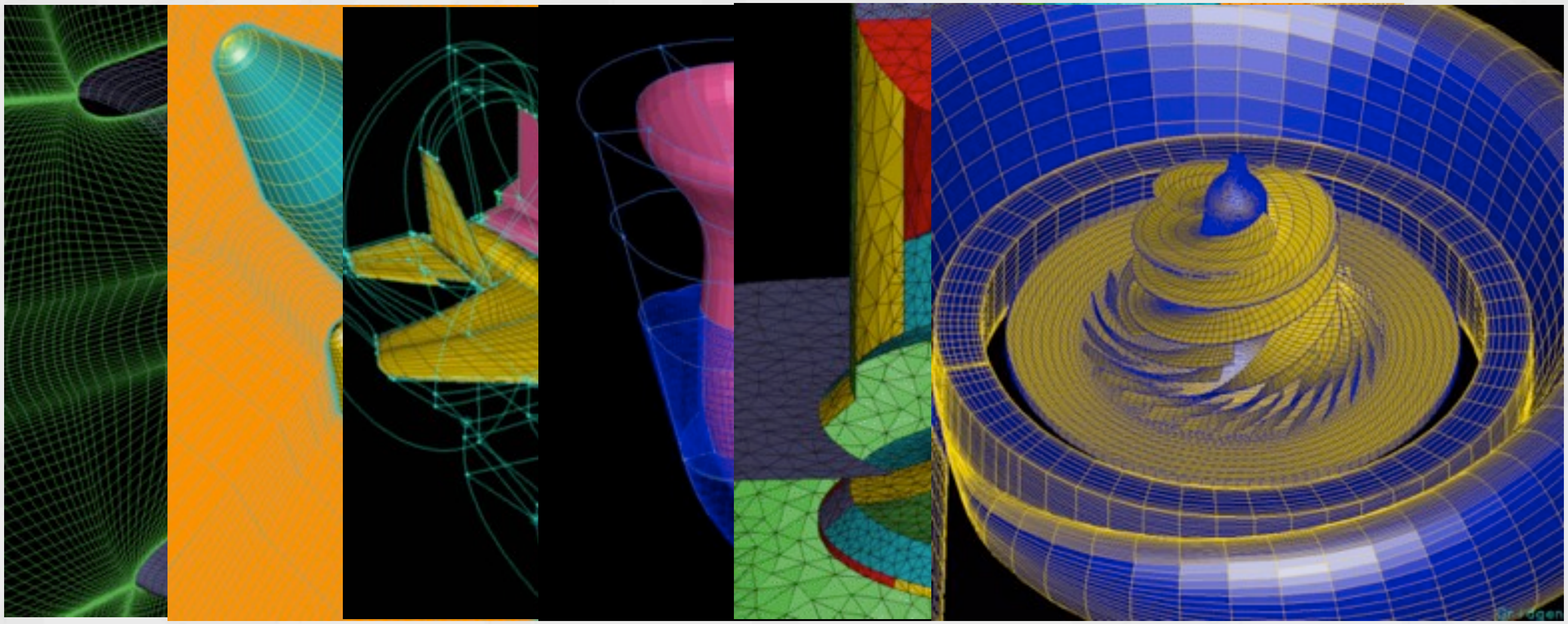
POINTWISE®
Reliable CFD Meshing

Gridgen Introduction

- Meshing software used by engineers and scientists worldwide since 1984.
- Complete toolkit for generating meshes with a variety of cell types (i.e., hexahedra, tetrahedra, prism).
- “Bottom-up” meshing approach (database-connector-domain-block).

POINTWISE®
Reliable CFD Meshing

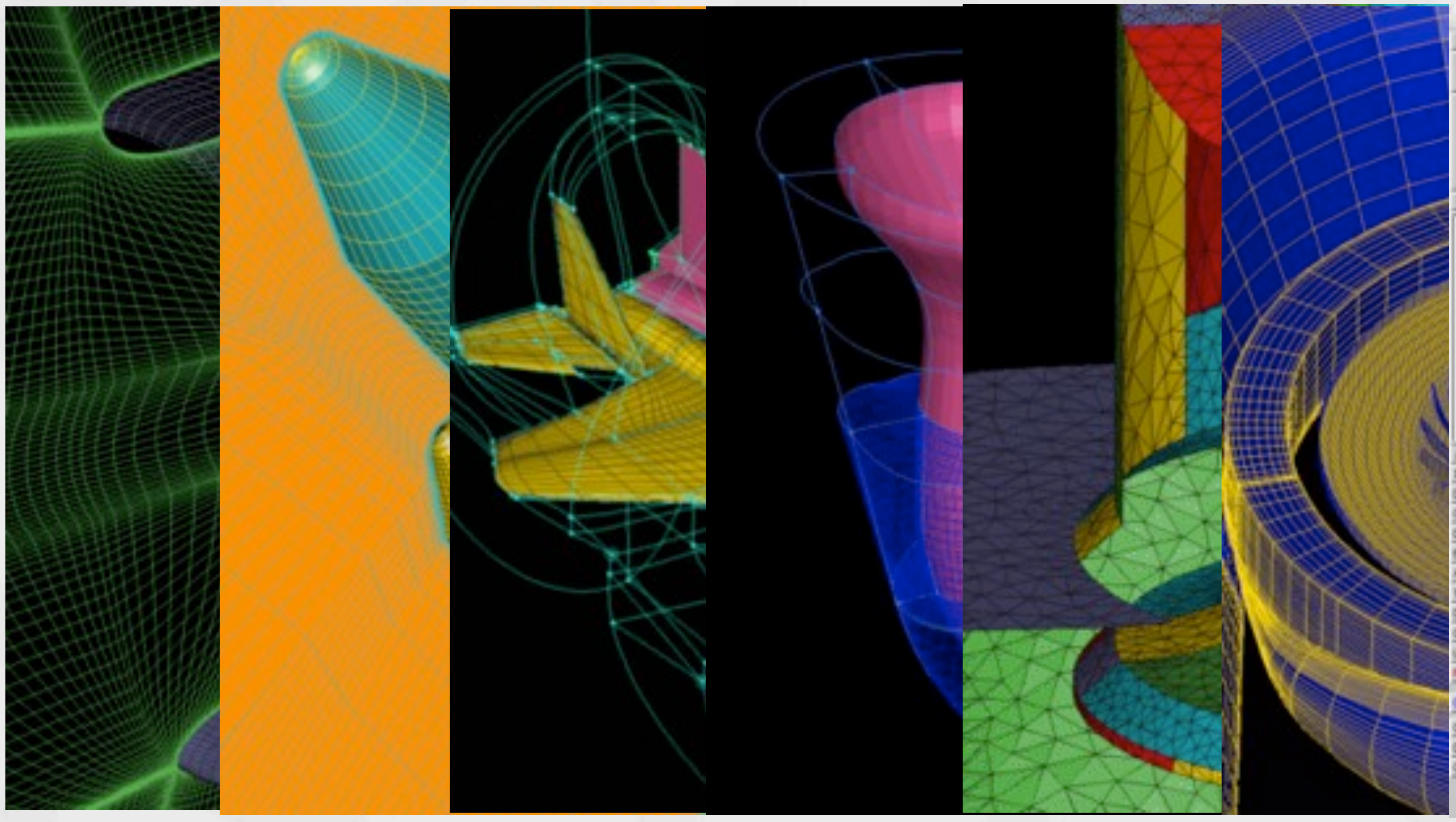
Gridgen Introduction

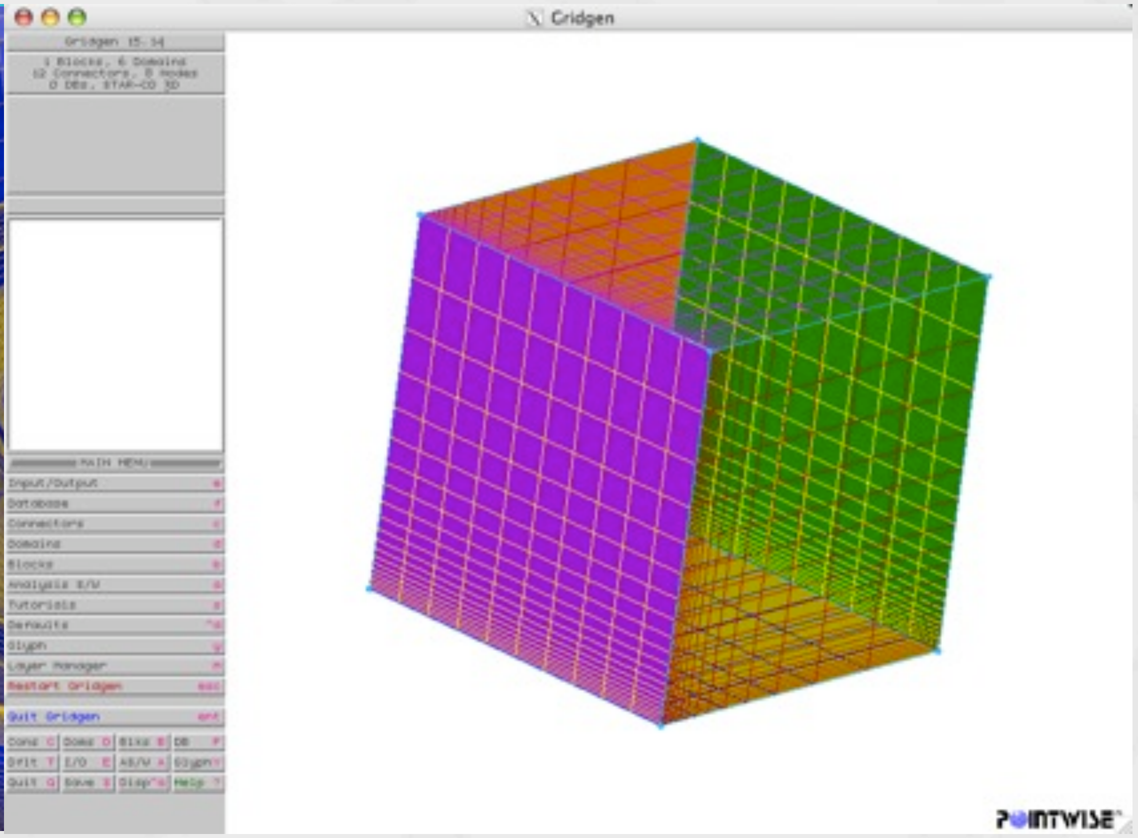


- Meshing software used by engineers and scientists worldwide since 1984.
- Complete toolkit for generating meshes with a variety of cell types (i.e., hexahedra, tetrahedra, prism).
- “Bottom-up” meshing approach (database-connector-domain-block).

POINTWISE®
Reliable CFD Meshing

Gridgen Introduction

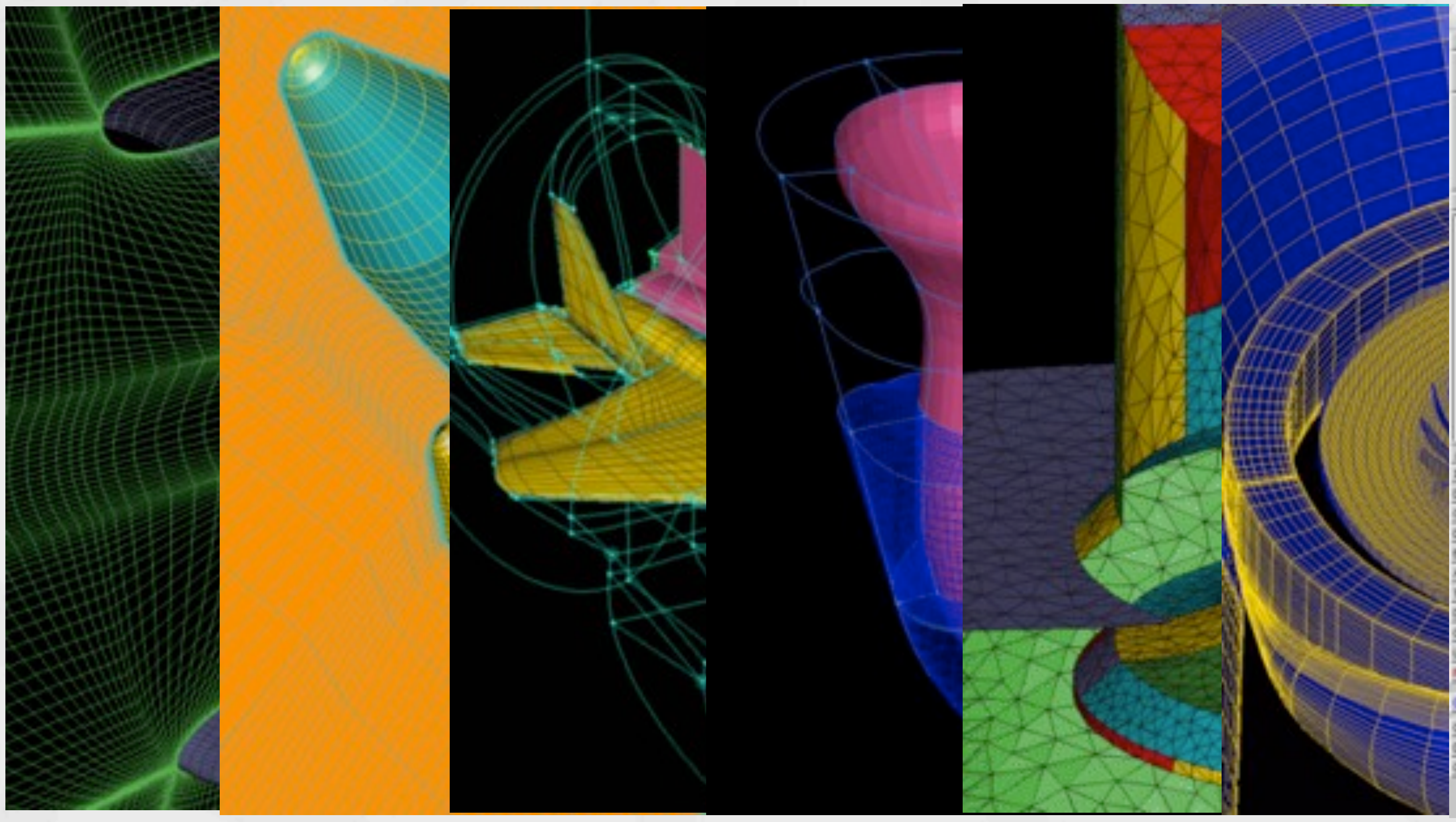


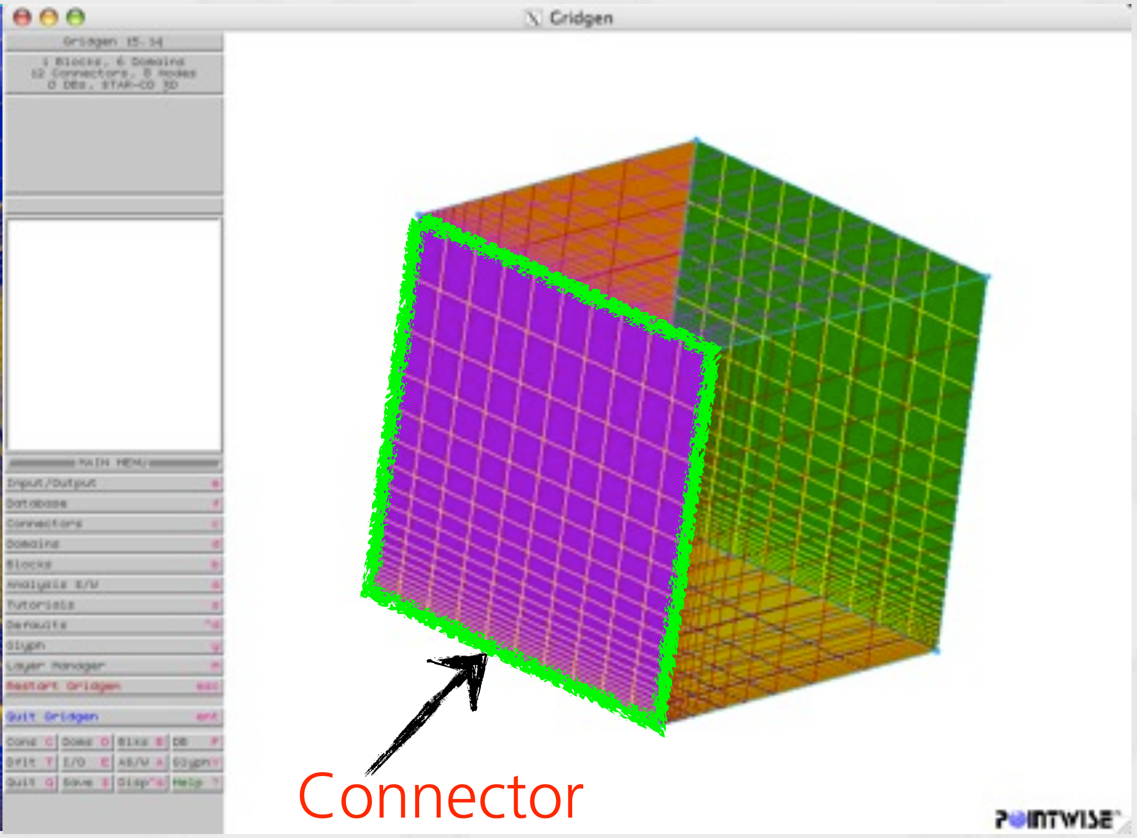


- Meshing software used by engineers and scientists worldwide since 1984.
- Complete toolkit for generating meshes with a variety of cell types (i.e., hexahedra, tetrahedra, prism).
- “Bottom-up” meshing approach (database-connector-domain-block).

POINTWISE®
Reliable CFD Meshing

Gridgen Introduction

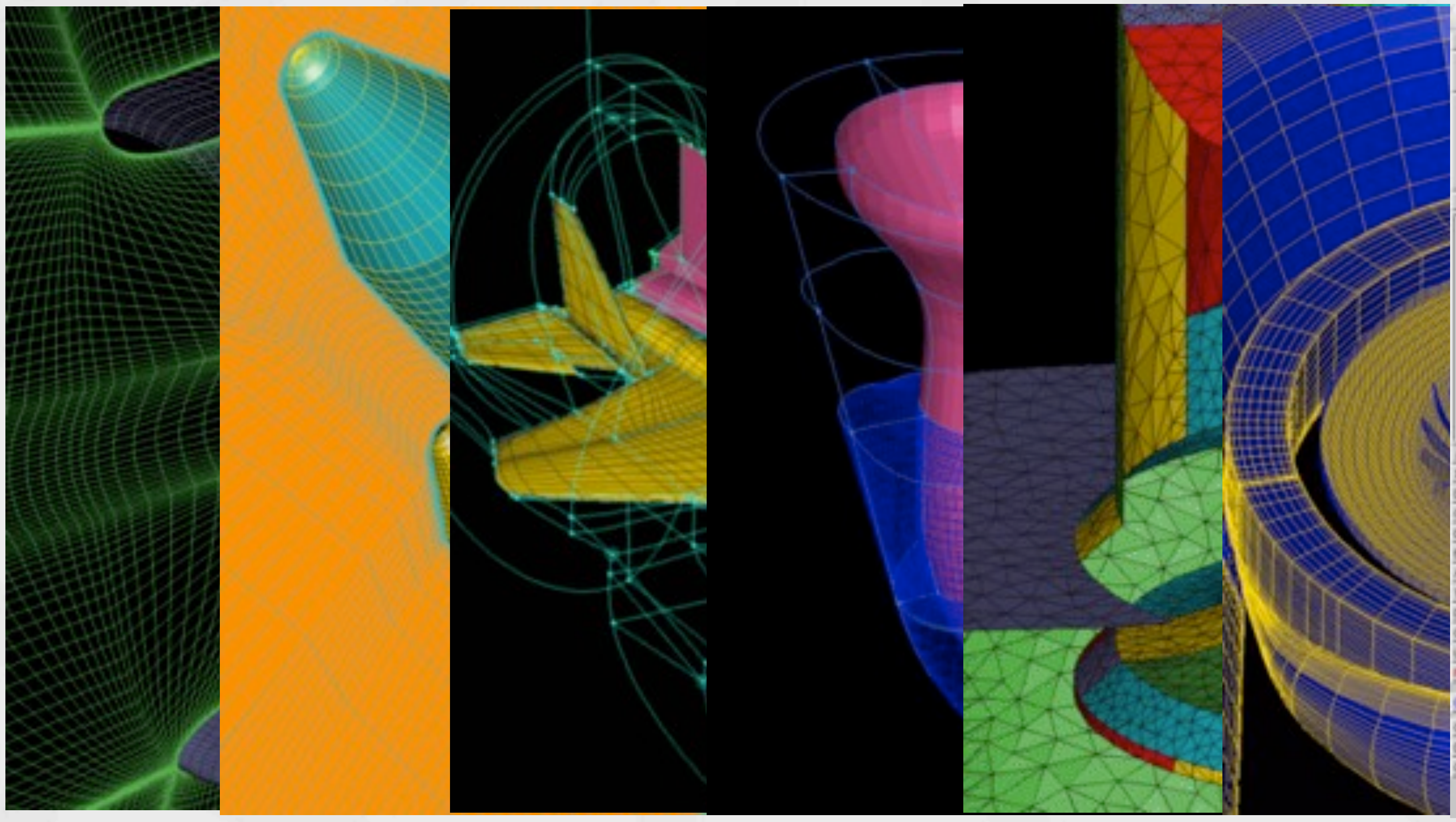


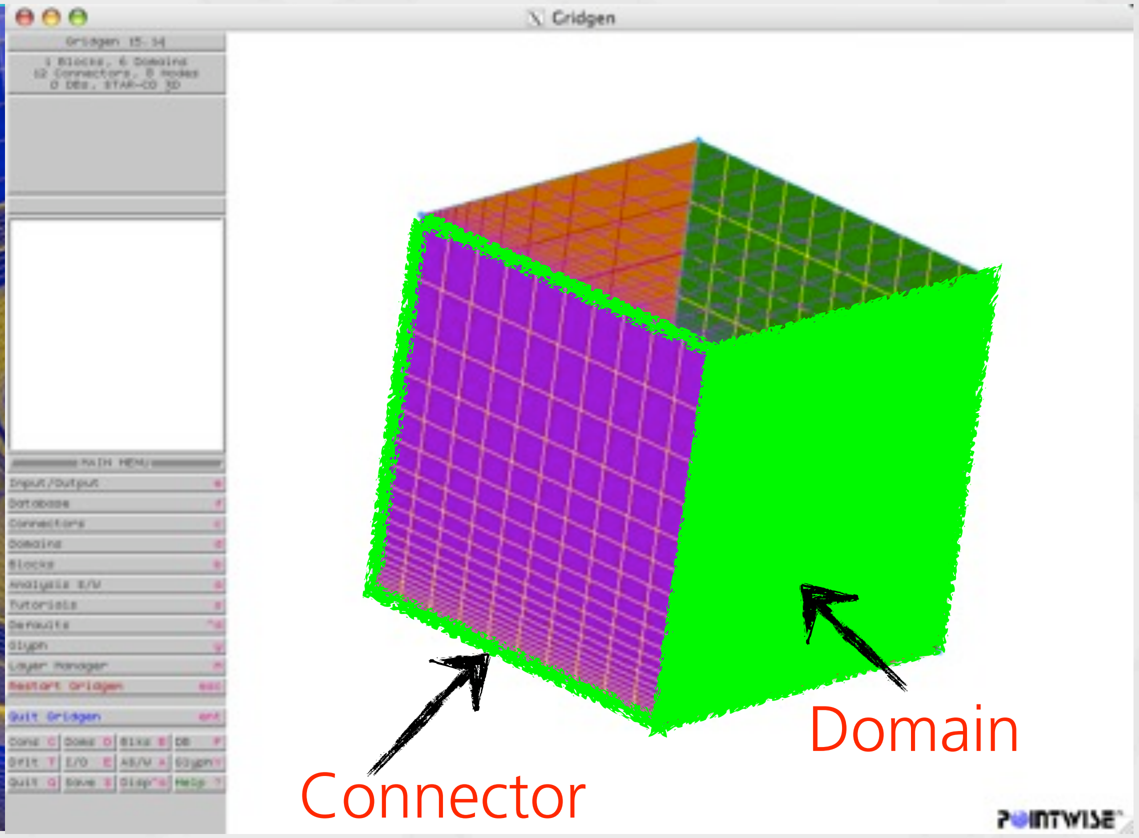


- Meshing software used by engineers and scientists worldwide since 1984.
- Complete toolkit for generating meshes with a variety of cell types (i.e., hexahedra, tetrahedra, prism).
- “Bottom-up” meshing approach (database-connector-domain-block).

POINTWISE
Reliable CFD Meshing

Gridgen Introduction



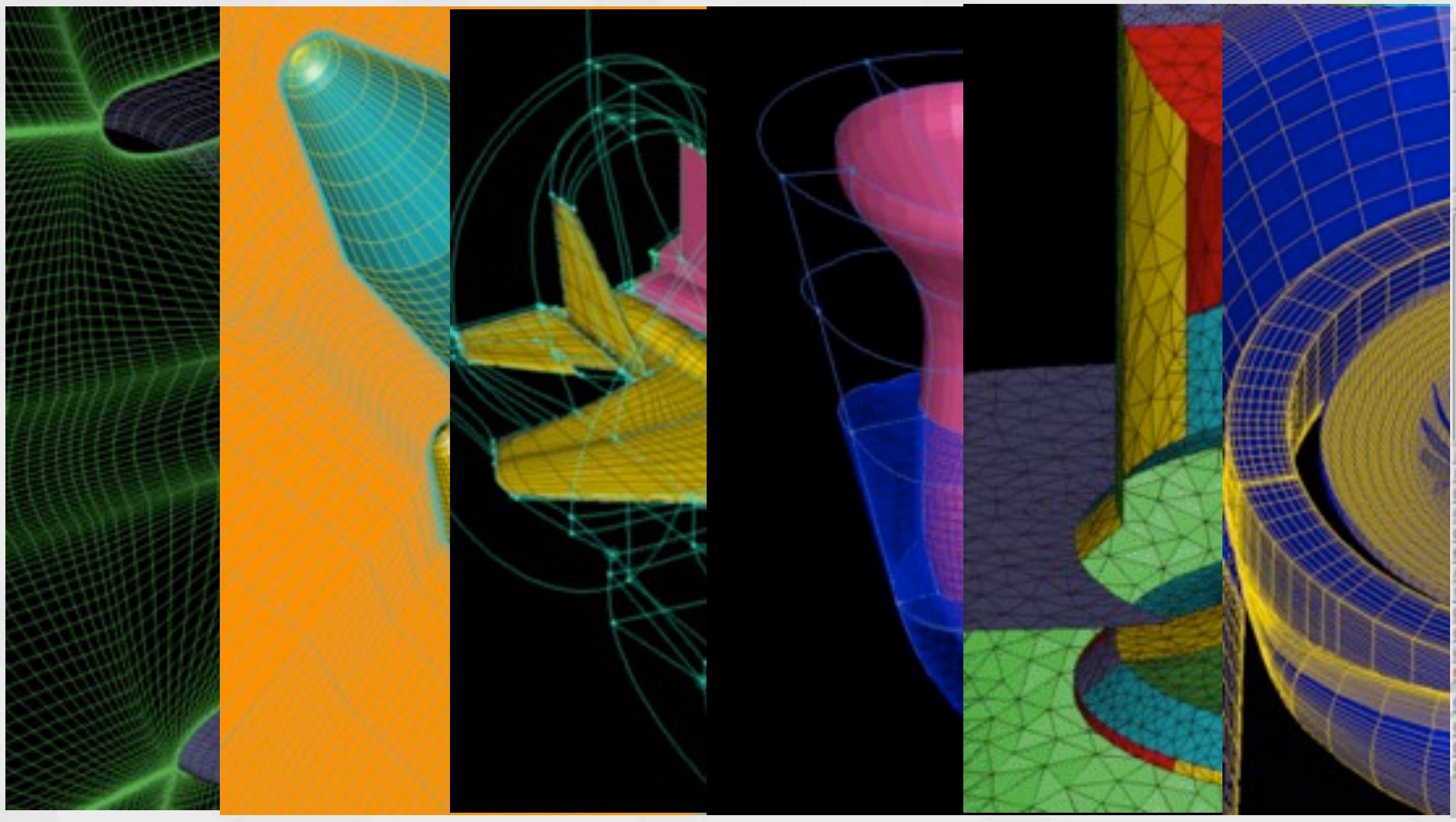


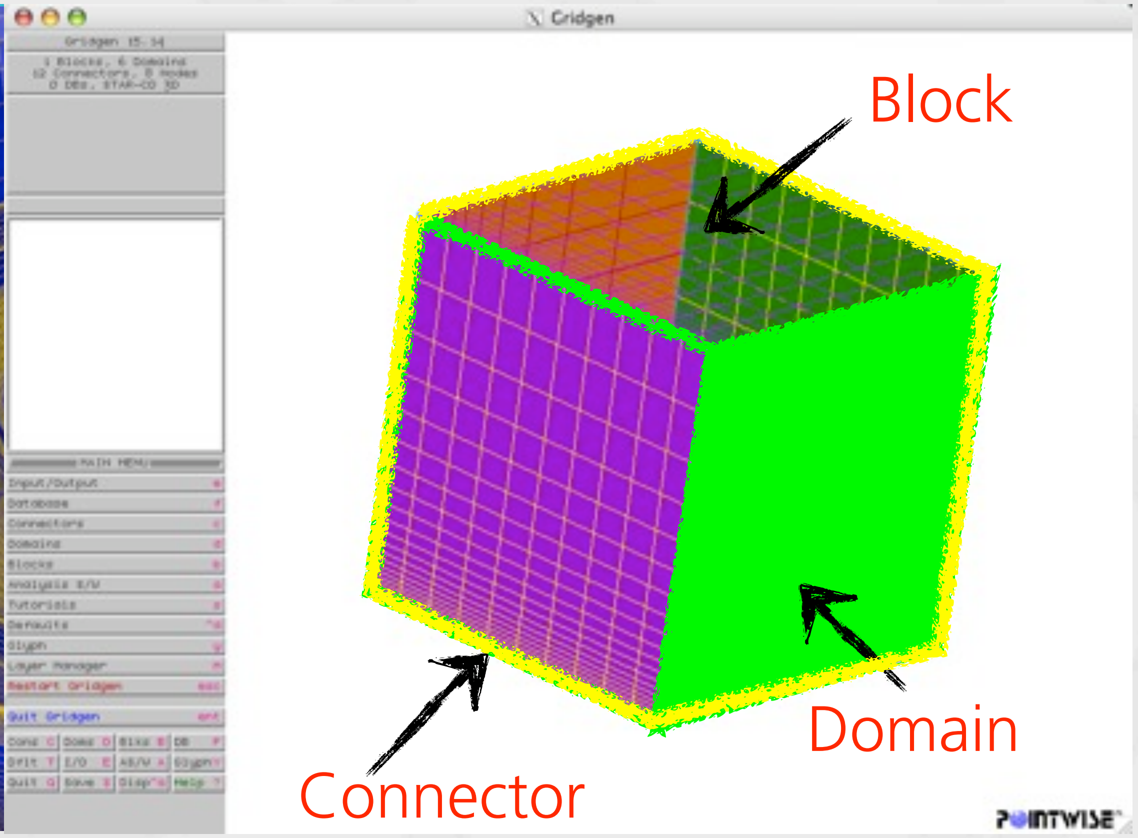
1 Blocks, 6 Domains
12 Connectors, 8 Nodes
0 DBs, STAR-CD 3D

- Meshing software used by engineers and scientists worldwide since 1984.
- Complete toolkit for generating meshes with a variety of cell types (i.e., hexahedra, tetrahedra, prism).
- “Bottom-up” meshing approach (database-connector-domain-block).

POINTWISE
Reliable CFD Meshing

Gridgen Introduction





```
Gridgen 15.14
1 Blocks, 6 Domains
12 Connectors, 8 Nodes
0 DBs, STAR-CD 3D
```

- Meshing software used by engineers and scientists worldwide since 1984.
- Complete toolkit for generating meshes with a variety of cell types (i.e., hexahedra, tetrahedra, prism).
- “Bottom-up” meshing approach (database-connector-domain-block).

POINTWISE® Reliable CFD Meshing

Glyph Scripting

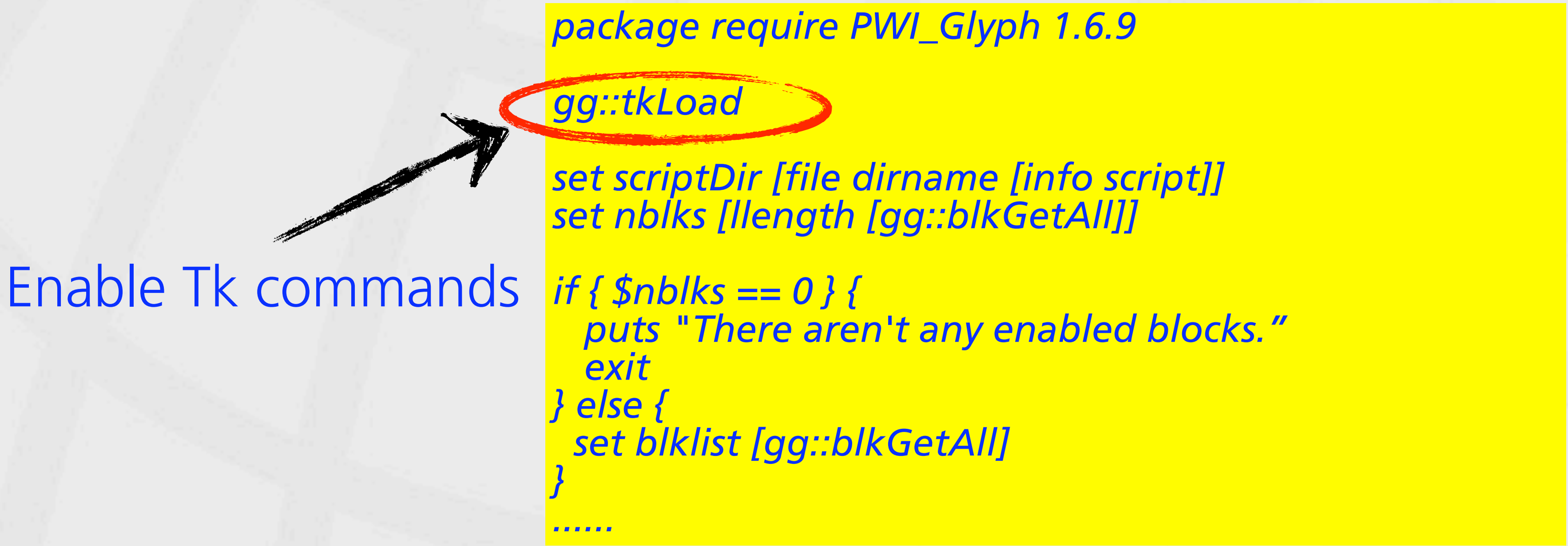
- Glyph (Tcl+Gridgen specific commands) provides a text-based, procedural interface to Gridgen’s features.

```
package require PWI_Glyph 1.6.9

gg::tkLoad

set scriptDir [file dirname [info script]]
set nblks [llength [gg::blkGetAll]]

if { $nblks == 0 } {
   puts "There aren't any enabled blocks.”
   exit
} else {
  set blklist [gg::blkGetAll]
}
......
```

- Glyph scripts can be executed in batch or Gridgen’s user interface.
- Glyph scripts are useful for:
  - Establishing preferred display states and default values.
  - Encapsulating repetitive tasks.
  - Developing specialized meshing applications.

Glyph Scripting

- Glyph (Tcl+Gridgen specific commands) provides a text-based, procedural interface to Gridgen's features.



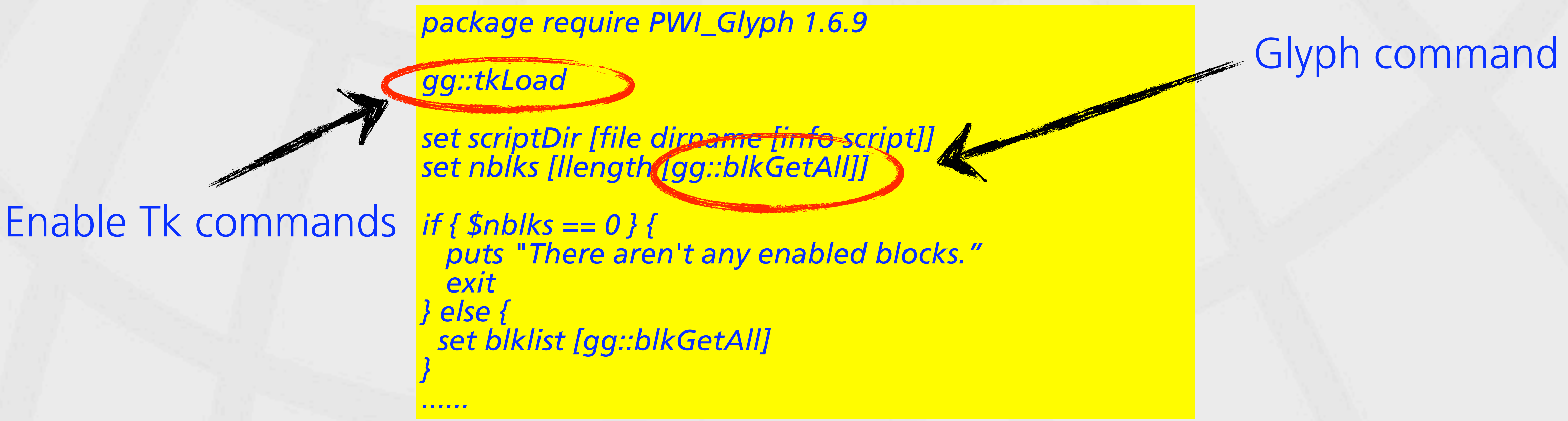
Enable Tk commands

```
package require PWI_Glyph 1.6.9

gg::tkLoad

set scriptDir [file dirname [info script]]
set nblks [llength [gg::blkGetAll]]

if { $nblks == 0 } {
   puts "There aren't any enabled blocks.”
   exit
} else {
  set blklist [gg::blkGetAll]
}
……
```

- Glyph scripts can be executed in batch or Gridgen's user interface.
- Glyph scripts are useful for:
  - Establishing preferred display states and default values.
  - Encapsulating repetitive tasks.
  - Developing specialized meshing applications.

Glyph Scripting

- Glyph (Tcl+Gridgen specific commands) provides a text-based, procedural interface to Gridgen's features.



```
package require PWI_Glyph 1.6.9

gg::tkLoad

set scriptDir [file dirname [info script]]
set nblks [llength [gg::blkGetAll]]

if { $nblks == 0 } {
   puts "There aren't any enabled blocks."
   exit
} else {
  set blklist [gg::blkGetAll]
}
......
```

Enable Tk commands

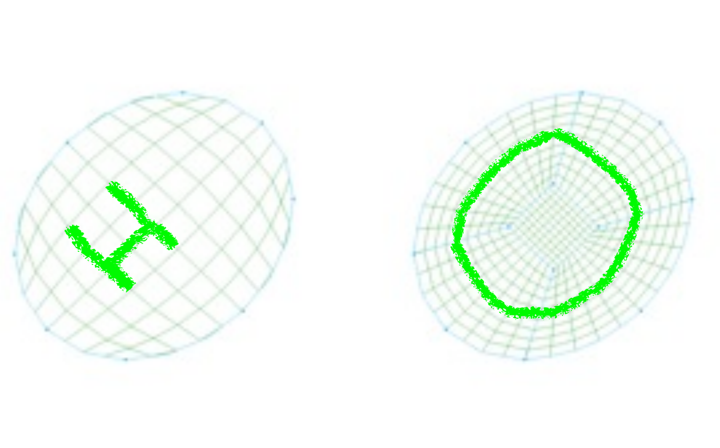
Glyph command

- Glyph scripts can be executed in batch or Gridgen's user interface.
- Glyph scripts are useful for:
  - Establishing preferred display states and default values.
  - Encapsulating repetitive tasks.
  - Developing specialized meshing applications.

POINTWISE
Reliable CFD Meshing

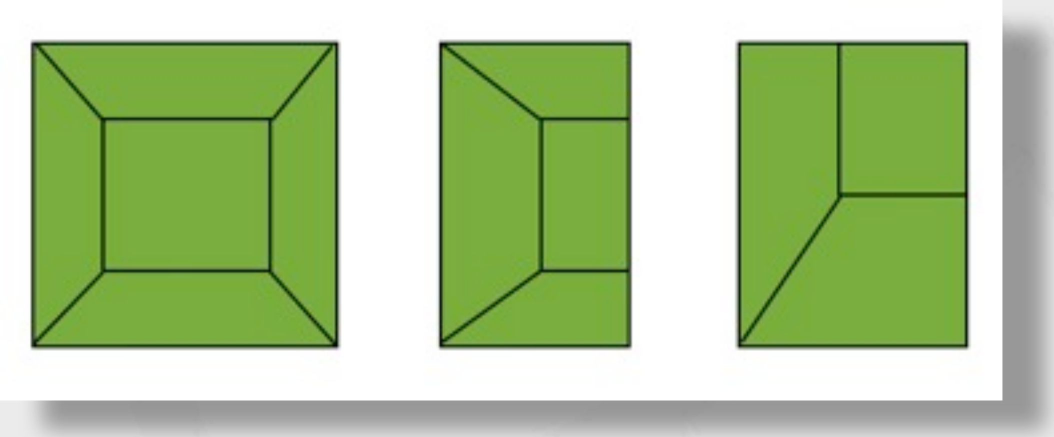
Butterfly (O–H) Topology

- What is an O-grid?
  - A series of blocks created with grid lines arranged into an “O” shape or a wrapping nature (i.e., “C” shape).



H-topology O-H topology

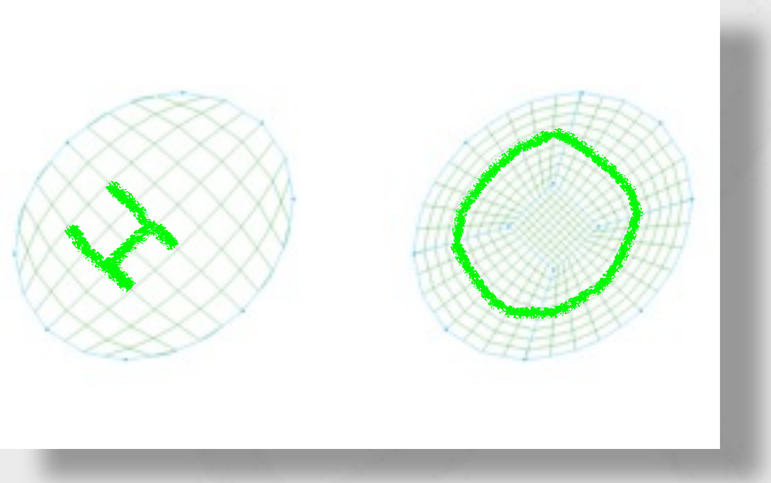
- What are the basic types?
  - O-H topology
  - C-H topology
  - L-H topology



O-H C-H L-H

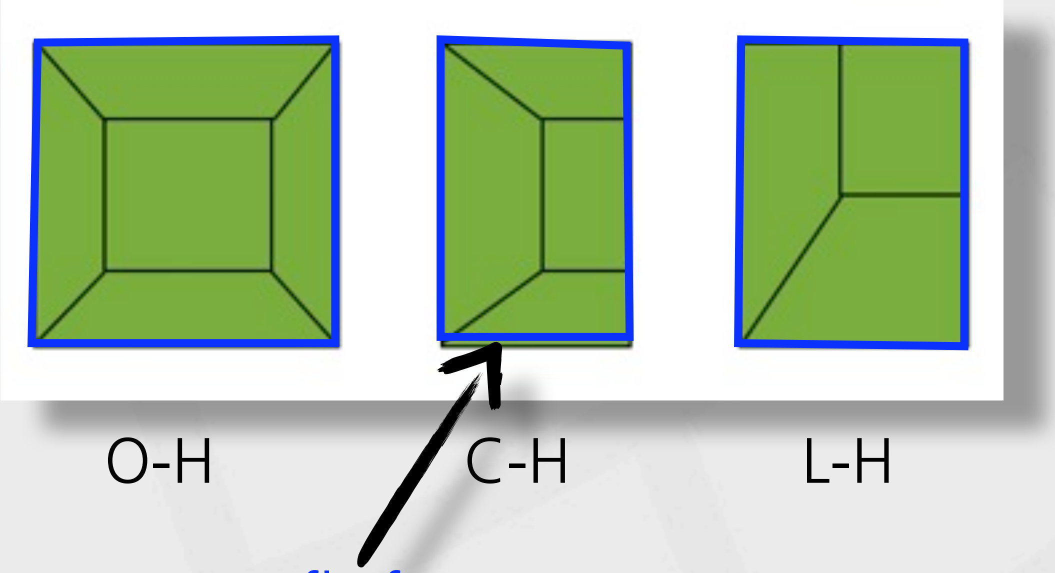
Butterfly (O–H) Topology

- What is an O-grid?
  - A series of blocks created with grid lines arranged into an “O” shape or a wrapping nature (i.e., “C” shape).



H-topology O-H topology

- What are the basic types?
  - O-H topology
  - C-H topology
  - L-H topology

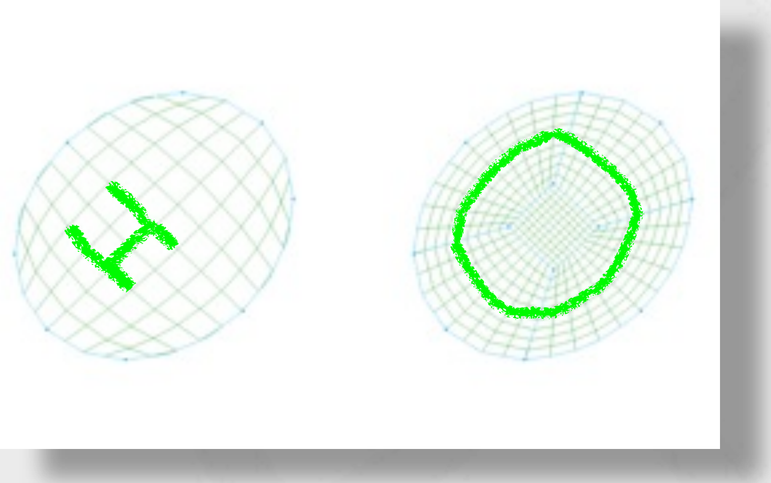


O-H C-H L-H

Butterfly face

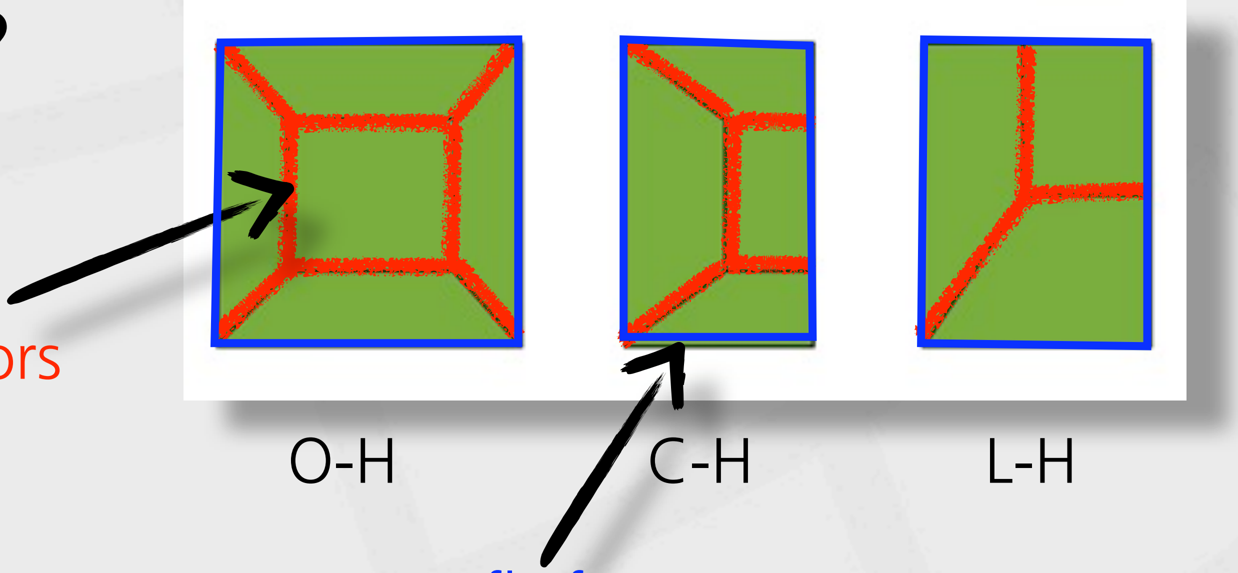
Butterfly (O–H) Topology

- What is an O-grid?
  - A series of blocks created with grid lines arranged into an “O” shape or a wrapping nature (i.e., “C” shape).



H-topology O-H topology

- What are the basic types?
  - O-H topology
  - C-H topology
  - L-H topology



Butterfly Connectors

O-H C-H L-H

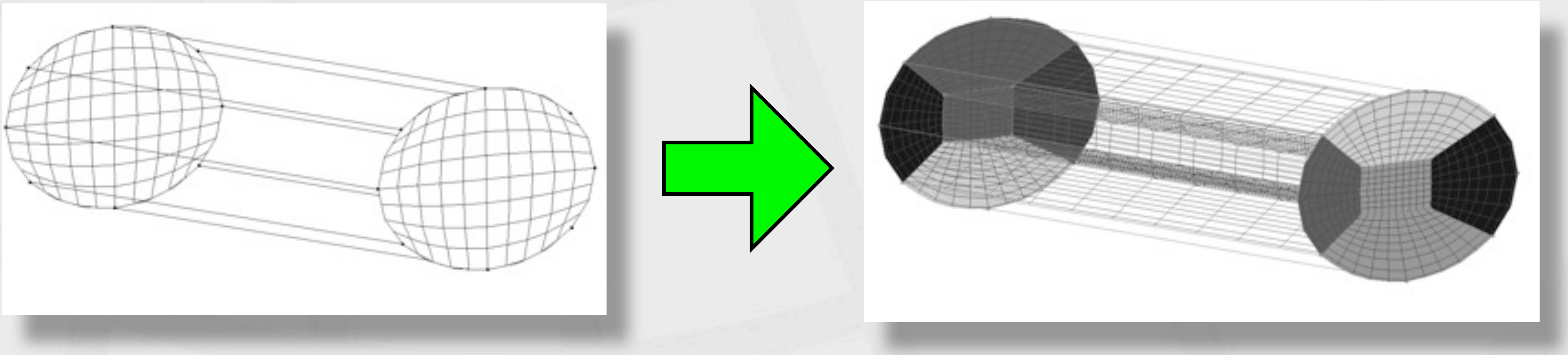
Butterfly face

Butterfly Topology (Cont ...)

- Why is an O-grid so useful?

H topology

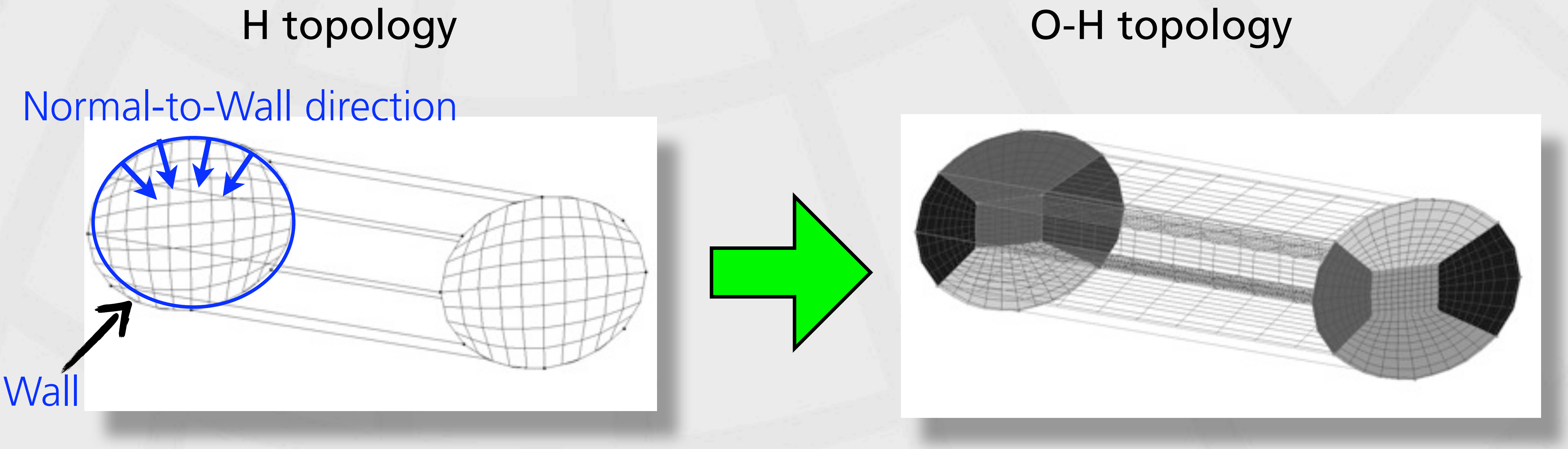
O-H topology



- Reduce skew where a block corner must lie on a continuous curve/surface.
- Improves efficiency of grid point clustering near walls.
- Resolve the boundary layer locally around solid bodies without unnecessarily increasing overall grid point count.

Butterfly Topology (Cont ...)

- Why is an O-grid so useful?



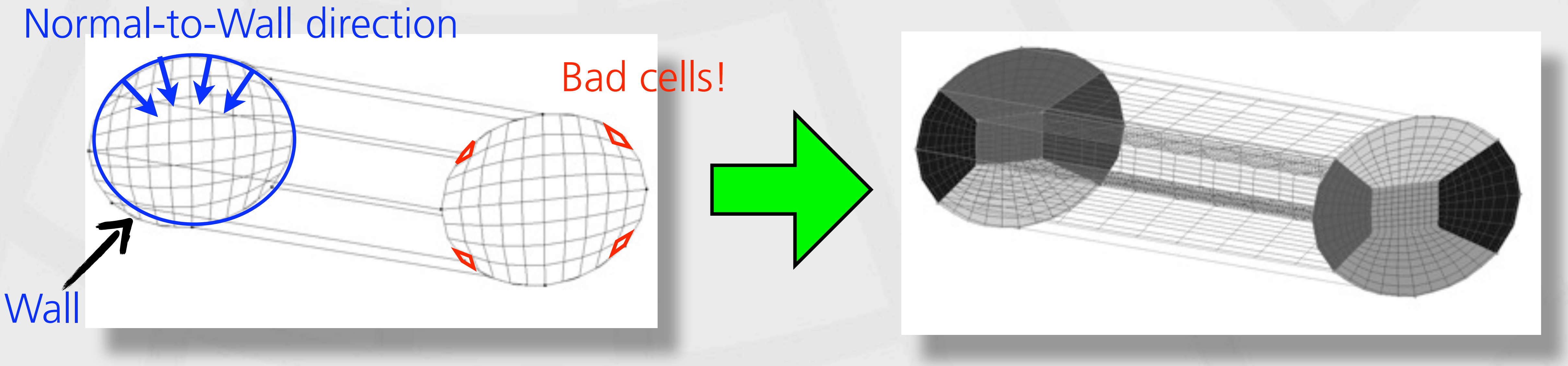
- Reduce skew where a block corner must lie on a continuous curve/surface.
- Improves efficiency of grid point clustering near walls.
- Resolve the boundary layer locally around solid bodies without unnecessarily increasing overall grid point count.

Butterfly Topology (Cont ...)

- Why is an O-grid so useful?

H topology

O-H topology



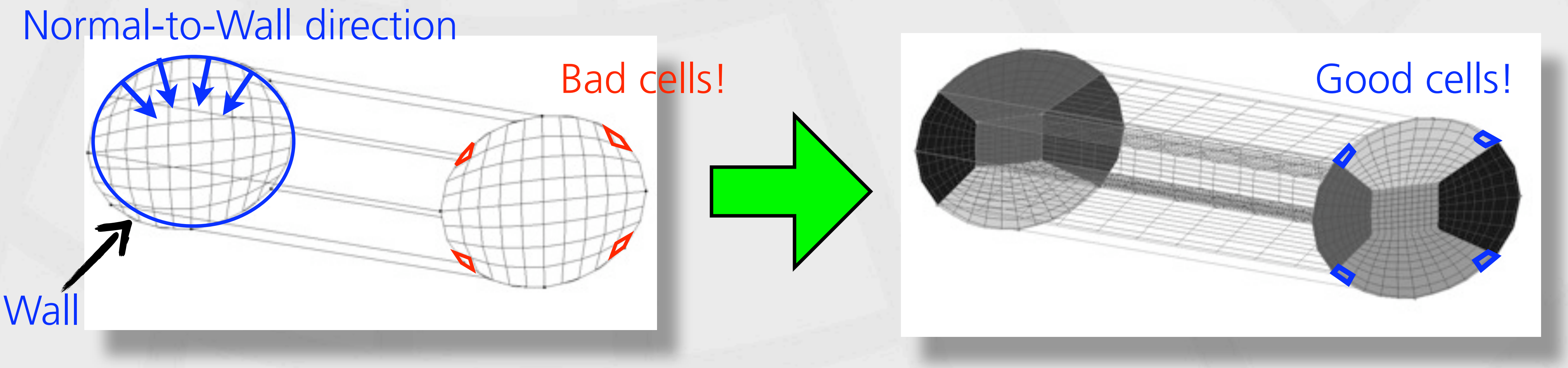
- Reduce skew where a block corner must lie on a continuous curve/surface.
- Improves efficiency of grid point clustering near walls.
- Resolve the boundary layer locally around solid bodies without unnecessarily increasing overall grid point count.

Butterfly Topology (Cont ...)

- Why is an O-grid so useful?

H topology

O-H topology



Normal-to-Wall direction

Bad cells!

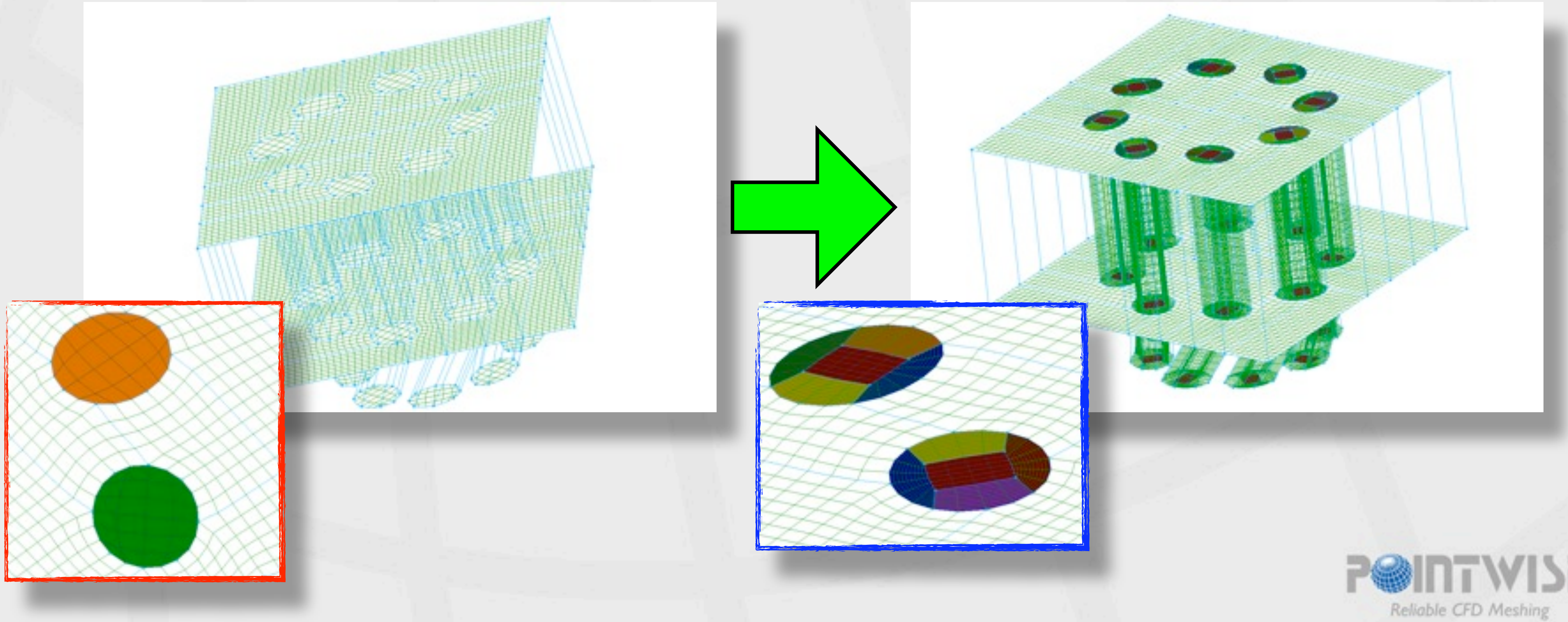
Wall

Good cells!

- Reduce skew where a block corner must lie on a continuous curve/surface.
- Improves efficiency of grid point clustering near walls.
- Resolve the boundary layer locally around solid bodies without unnecessarily increasing overall grid point count.

Background (Cont ...)

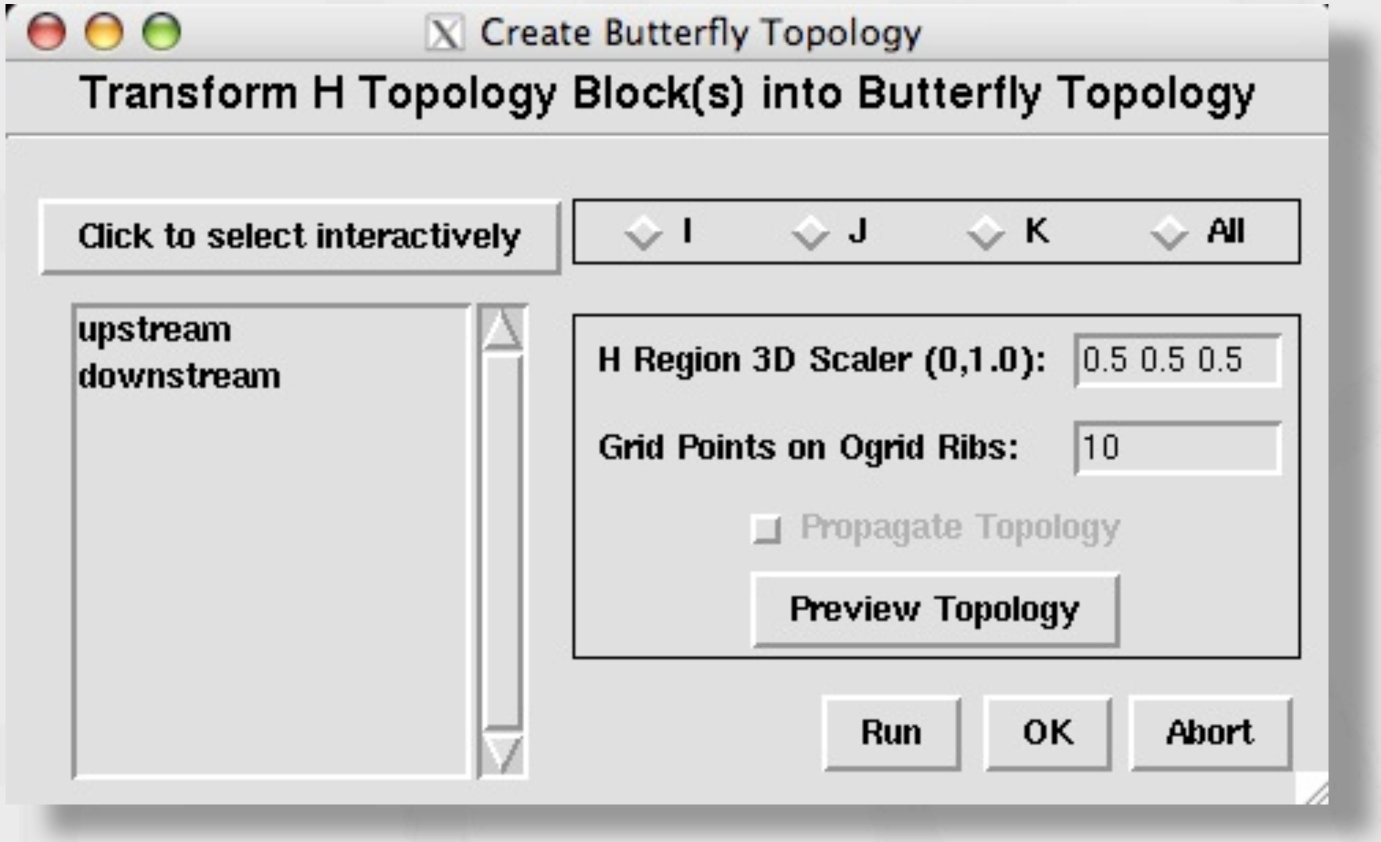
- Why do we need a script?
  - Gridgen does not have automatic O-H grid creation capability.
  - Interactively changing grid topologies can take hours.
  - This is “one of the most important features” of our major competitor.



Script Overview

- Written in Tcl/Tk 8.3 and Glyph 1.0.
- Approximately 4,000 lines and 50 procedures.
- Main features:
  - Quickly transform H-blocks to O-H topology blocks (1 min vs. 1 hour or more).
  - Propagate new topology in a series of H-blocks regardless of their orientation.
  - ★ Maintain connector distributions in the propagating direction.
  - ★ Allow non-homogeneous scaling of the new O/H blocks in three directions (I/J/K).
  - ★ Allow butterfly faces consisting of multiple domains.
  - ★ Allow butterfly faces with high curvature and/or slope discontinuity.

User Interface



Create Butterfly Topology

Transform H Topology Block(s) into Butterfly Topology

upstream
downstream

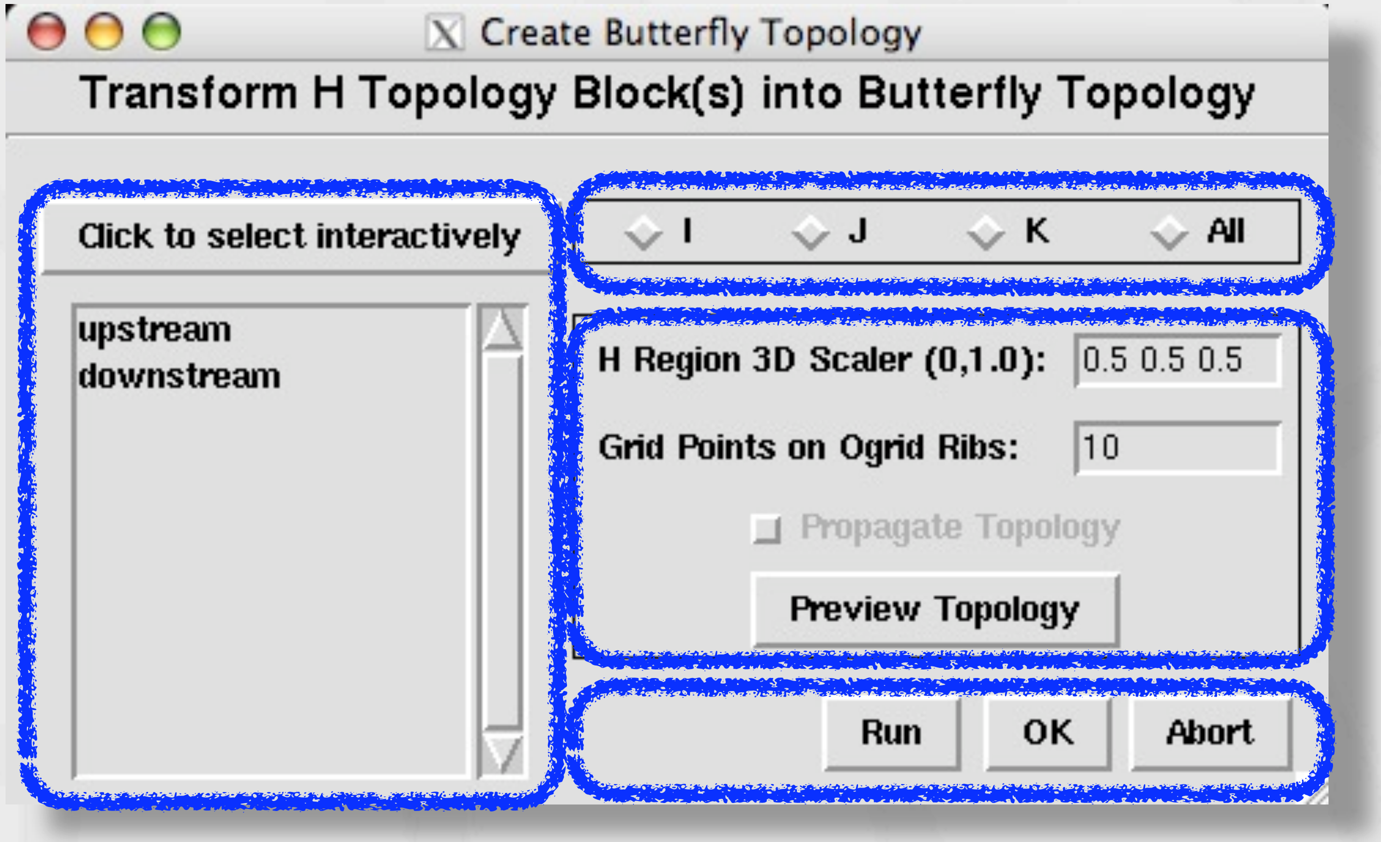
◇ I ◇ J ◇ K ◇ All

H Region 3D Scaler (0,1.0): 0.5 0.5 0.5

Grid Points on Ogrid Ribs: 10

□ Propagate Topology

User Interface



Transform H Topology Block(s) into Butterfly Topology

upstream
downstream

H Region 3D Scaler (0,1.0): 0.5 0.5 0.5

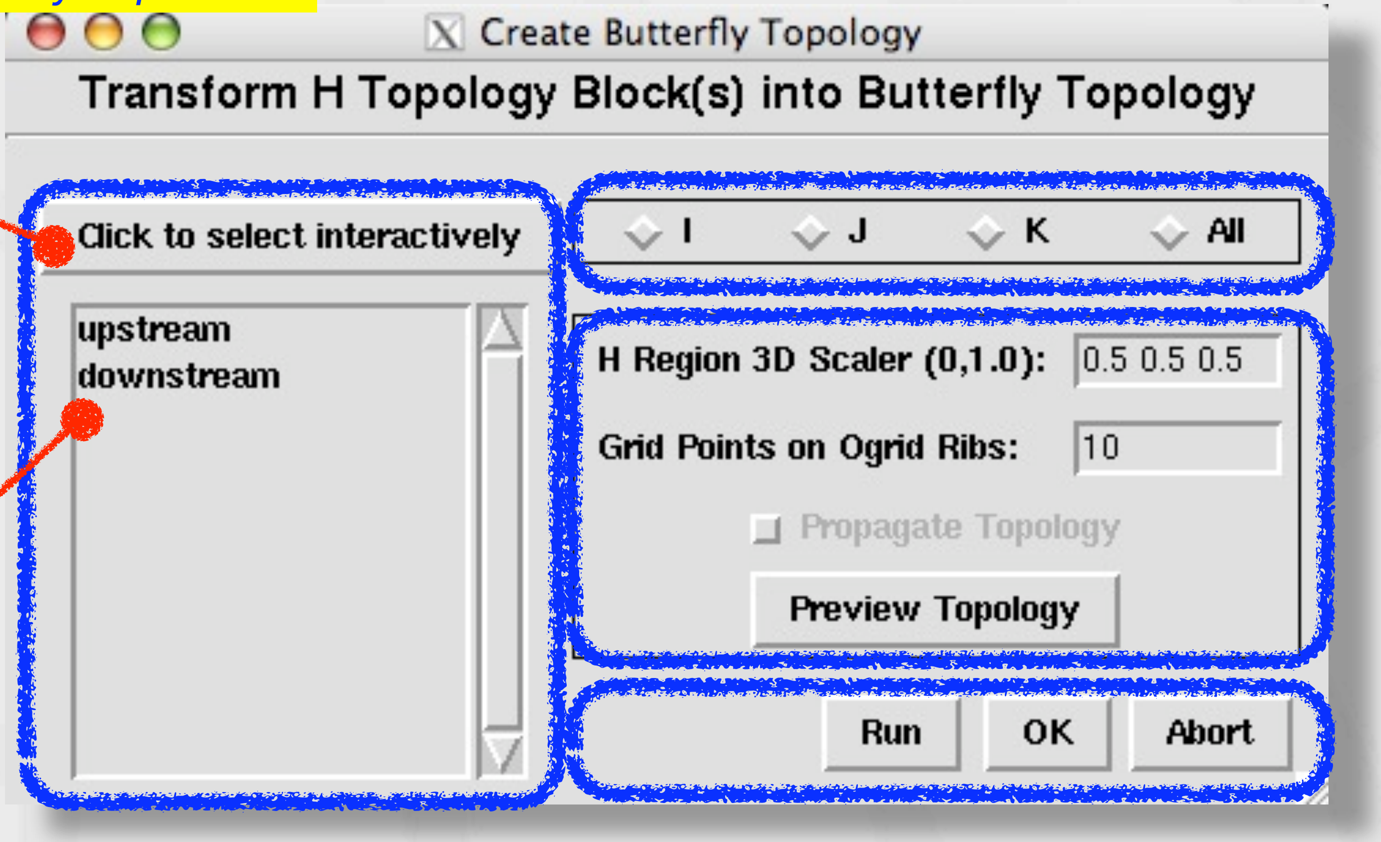
Grid Points on Ogrid Ribs: 10

□ Propagate Topology

User Interface

```
pack [button .right.select \
-text "Click to select interactively" \
-command select] -side top -pady 8 -padx 1
```

Interactive block selection



Transform H Topology Block(s) into Butterfly Topology

upstream
downstream

H Region 3D Scaler (0,1.0): 0.5 0.5 0.5

Grid Points on Ogrid Ribs: 10

Block selection via list box

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

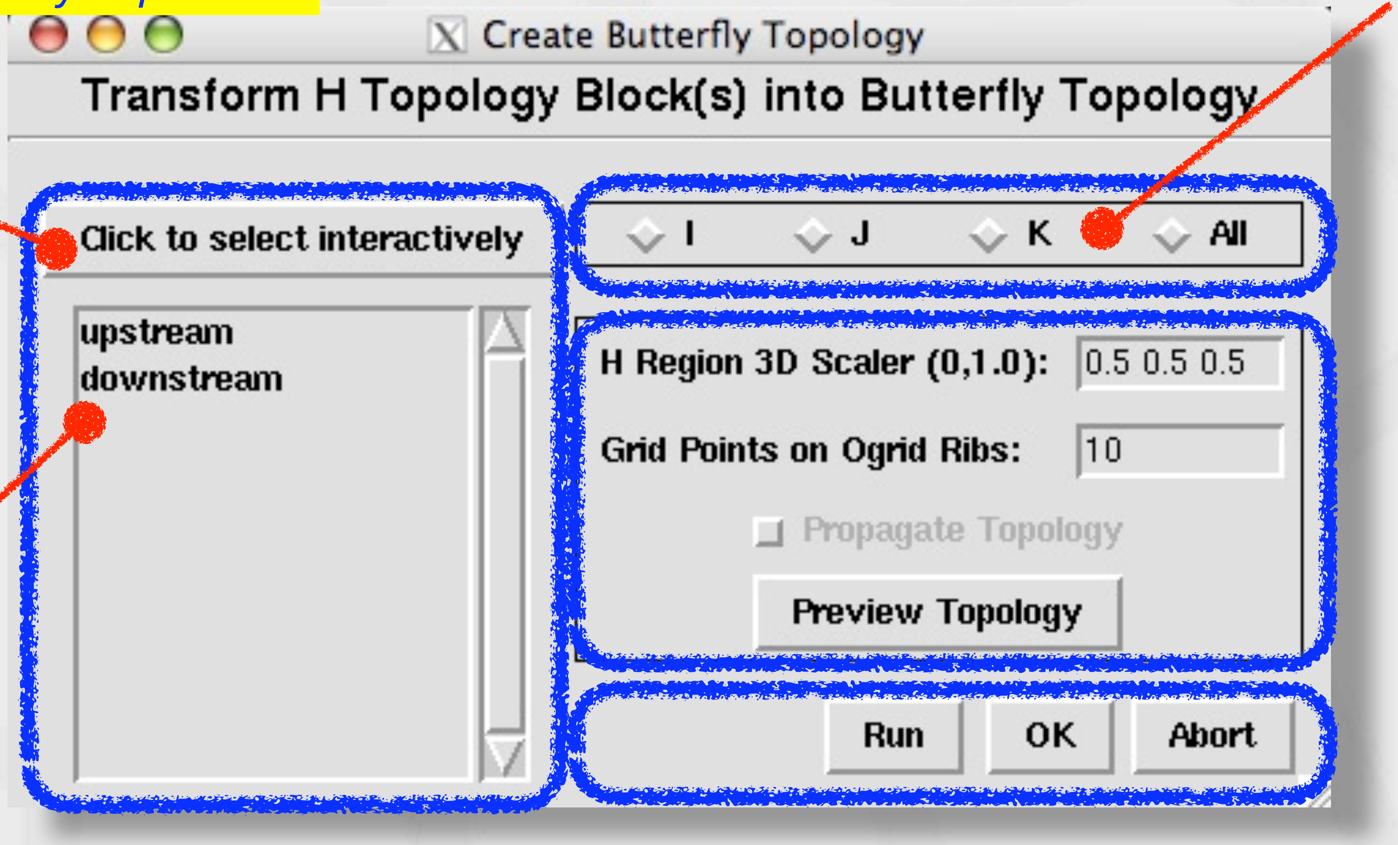
User Interface

```
pack [radiobutton .left.dir.i -text "I" -command { \
        global Direction Duplicate_Direction; \
        if { [string compare $Direction \
          $Duplicate_Direction] != 0 } { SwitchMode 1 }; \
        set Duplicate_Direction $Direction } \
        -variable Direction -value I] -side left -expand 1
```

```
pack [button .right.select \
-text "Click to select interactively" \
-command select] -side top -pady 8 -padx 1
```

Interactive block selection

Propagating direction



Create Butterfly Topology

Transform H Topology Block(s) into Butterfly Topology

Click to select interactively

upstream
downstream

◇ I ◇ J ◇ K ◇ All

H Region 3D Scaler (0,1.0): 0.5 0.5 0.5

Grid Points on Ogrid Ribs: 10

□ Propagate Topology

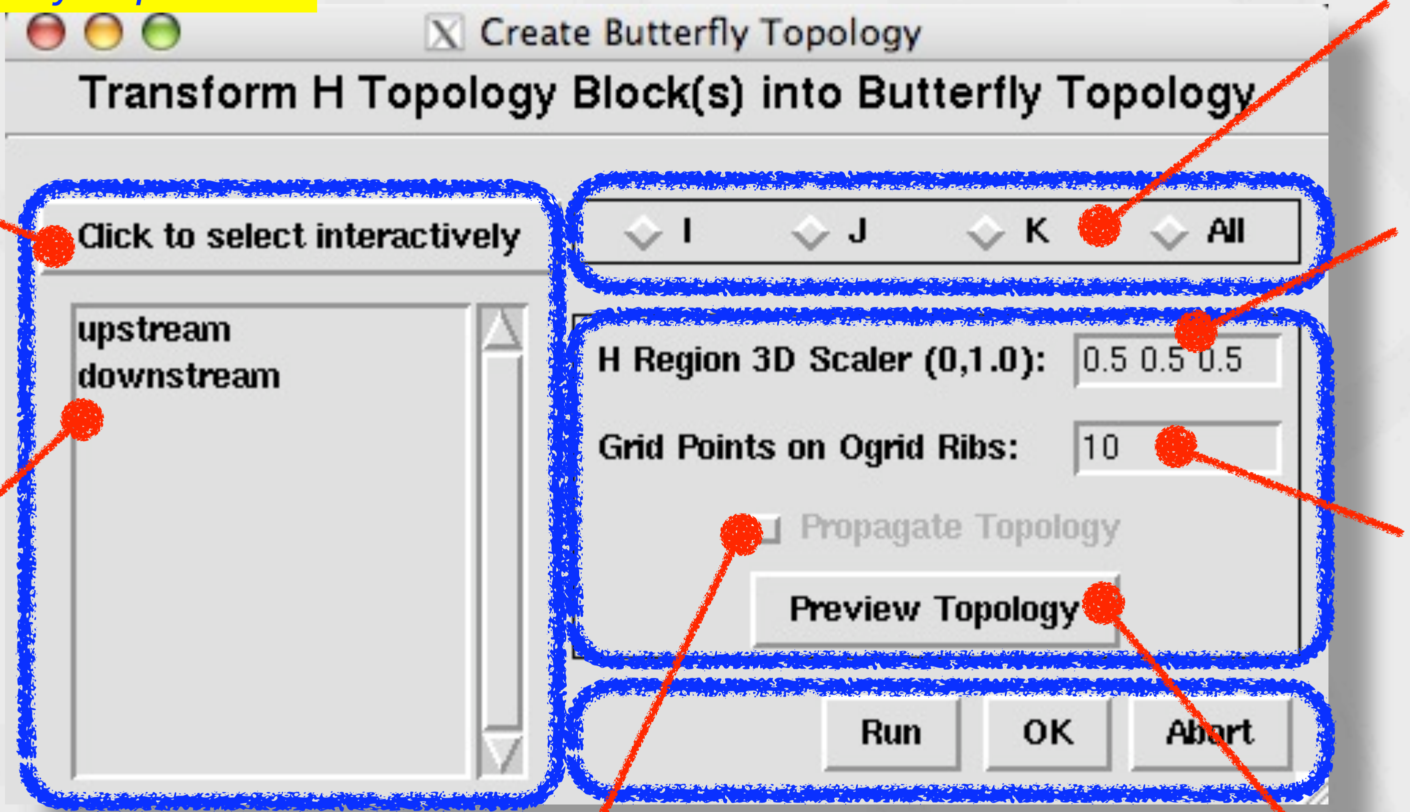
Block selection via list box

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

User Interface

```
pack [radiobutton .left.dir.i -text "I" -command { \
        global Direction Duplicate_Direction; \
        if { [string compare $Direction \
          $Duplicate_Direction] != 0 } { SwitchMode 1 }; \
        set Duplicate_Direction $Direction } \
        -variable Direction -value I] -side left -expand 1
```

```
pack [button .right.select \
-text "Click to select interactively" \
-command select] -side top -pady 8 -padx 1
```


Create Butterfly Topology
Transform H Topology Block(s) into Butterfly Topology
Click to select interactively
upstream
downstream
◇ I
◇ J
◇ K
◇ All
□ Propagate Topology
Preview Topology
Run
OK
Abort

H Region 3D Scaler (0,1.0): 0.5 0.5 0.5

Grid Points on Ogrid Ribs: 10

Interactive block selection

Propagating direction

3D scaler

```
pack [makeInputField \
.left.values dist \
 "H Region 3D Scaler (0,1.0):"\
oScaleFac] \
-fill x -padx 2 -pady 4
```

O-grid refining

```
 pack [makeInputField \
.left.values pts \
 "Grid Points on Ogrid Ribs:"\
oDimension] \
-fill x -padx 2 -pady 4
```

Block selection via list box

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

Forcing propagation check

```
checkbutton .left.values.propagate -text \
                  "Propagate Topology" -variable Propagate \
                  -command {Redraw}
```

New topology preview

```
button .left.values.preview \
         -text "Preview Topology" \
         -command {global locatorH_prop; \
         Redraw; set locatorH_prop {}}
```

Main Workflow

- Validate user input.
- Obtain the propagating block list.
- Determine which domains will be turned into "butterfly domains" and which will be kept.
- Locate the center domain on each butterfly face and create it butterfly connectors.
- Create new internal connectors in the propagating direction.
- If more than one block is selected, make sure no conflicts occur at the block interface.
- Match up the distributions of new connectors with their counterparts in the original blocks.
- Assemble the butterfly and internal domains.
- Assemble the new O-H blocks.

#1: User Input Diagnostics

- Is the scaling factor valid?
  - The three elements, S1, S2 and S3, must be in the range of (0, 1).
- Is the grid point number valid?
- Is the propagating block list valid?
  - The blocks have to be connected one to another.
  - The blocks have to share full faces in the propagating direction.
  - There is no duplicated blocks in the list.
- 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.

```
if { $oScale_1 > 1.0 || $oScale_2 > 1.0 || $oScale_3 > 1.0 \
    || $oScale_1 < 0.0 || $oScale_2 < 0.0 || $oScale_3 < 0.0 } {
    ErrorMsg "Invalid scaling factor input!"
    return
 }
```

```
proc getPropagatedBlockList { blk dir } {
        global Propagate
        # If Propagate checkbox is not checked, the original
        # selected block will be returned immediately.
        if { $Propagate == 0 } {
           return $blk
        } else {
          lappend blkList "$blk $dir"
          for { set i 0 } { $i < [llength $blkList] } { incr i } {
            foreach n [getAdjacentBlocks [lindex \
                              [lindex $blkList $i] 0] \
                              [lindex [lindex $blkList $i] 1]] {
              if { [lsearch $blkList $n] == -1 } {
                 lappend blkList "$n"
              }
            }
          }
  return $blkList
}
```

#2: Universal Indexing

- Operates independently to (I, J, K) once it is defined.
- All the domain and connector indices 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)

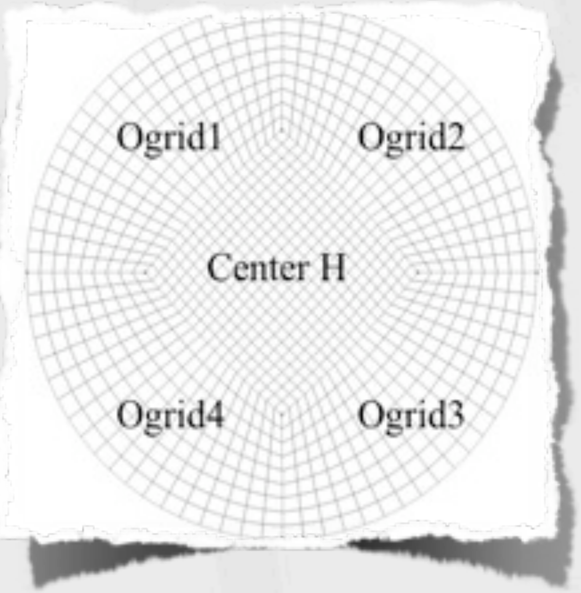


Table. 1 Example of grid entity indices

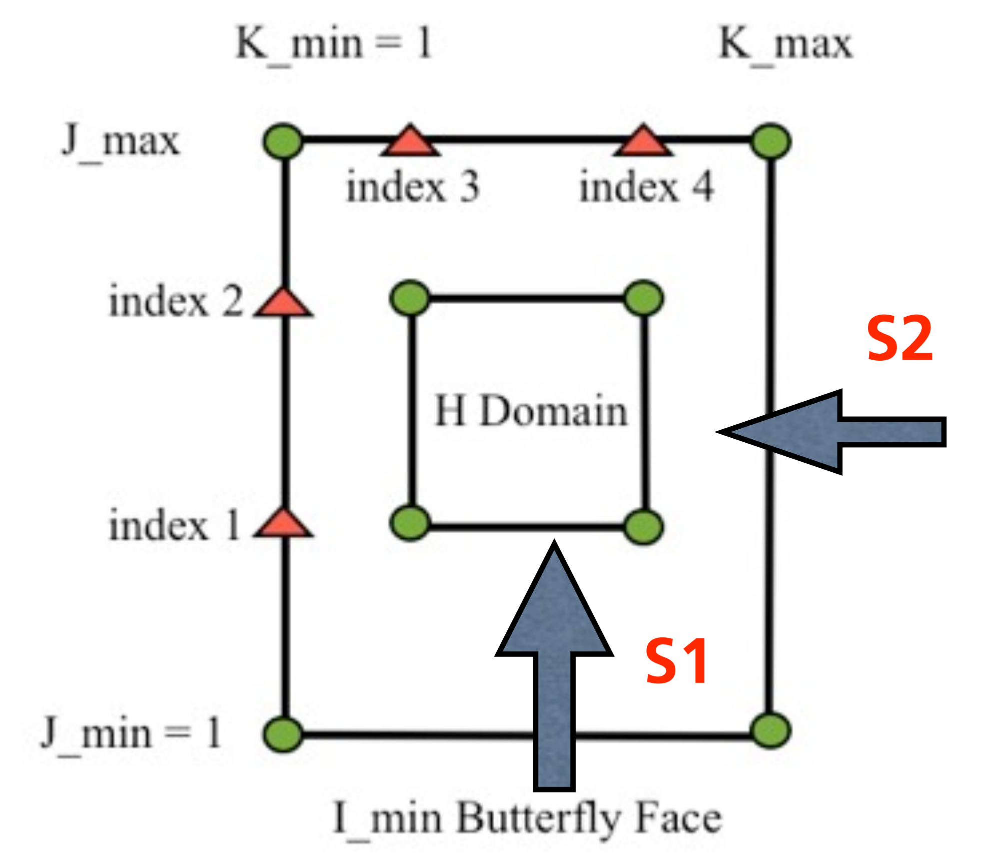
Block Name	center	ogrid1
Domain Index	(center, ind3_min), (center, ind3_max), (center, ind1_min), (center, ind1_max), (center, ind2_min), (center, ind2_max)	(ogrid1, ind3_min), (ogrid1, ind3_max), (original face 1), (center, ind2_min), (corner1), (corner2)
Connector Index	(ind2_min, ind3_min), (ind1_max,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

```
gg::blkBegin -type STRUCTURED
  gg::faceBegin
    gg::faceAddDom $doms(center,ind3_min)
  gg::faceEnd
  gg::faceBegin
    gg::faceAddDom $doms(center,ind3_max)
  gg::faceEnd
  gg::faceBegin
    gg::faceAddDom $doms(center,ind1_min)
  gg::faceEnd
  gg::faceBegin
    gg::faceAddDom $doms(center,ind1_max)
  gg::faceEnd
  gg::faceBegin
    gg::faceAddDom $doms(center,ind2_min)
  gg::faceEnd
  gg::faceBegin
    gg::faceAddDom $doms(center,ind2_max)
  gg::faceEnd
 set blks(center) [gg::blkEnd]
```

#3: H Domain Locator

- 3-D scaler implementation
  - (S1, S2, S3)
  - (S1) Length between index 1 and index 2 is approximately the scaled length in J direction.
  - (S2) Length between index 3 and index 4 is approximately the scaled length in K direction.
- Pinpoint the 4 H domain corners.
- Create butterfly connectors.
- Assemble H and O domains.
- Shape information check

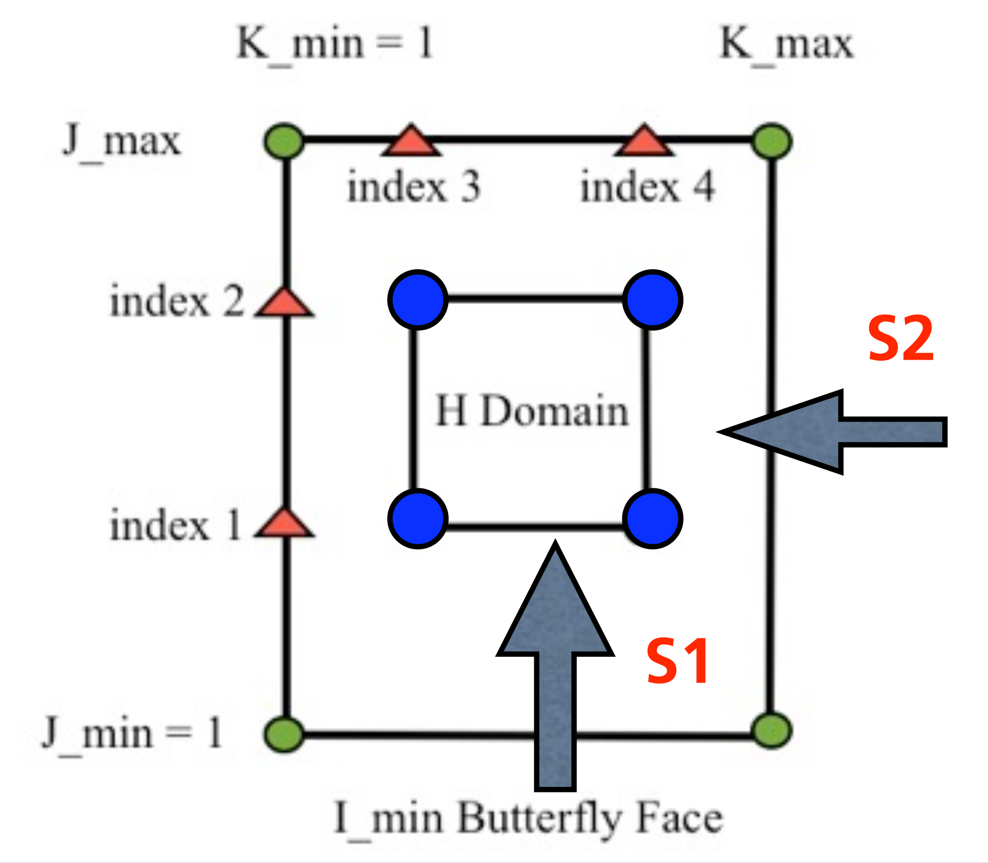
```
gg::dbImport $butterflyDBFile -style PLOT3D \
                    -format ASCII -precision DOUBLE

gg::domProject $butterflyDomList -type CLOSEST_PT \
                    -maintain_linkage
```



```
foreach end {ind3_min ind3_max} {
 foreach beg {ind1_min ind1_max} {
  set pt0 [gg::conGetPt $con($beg,$end) -arc 0]
  set pt1 [gg::conGetPt $con($beg,$end) -arc 1]
  if { [catch {gg::conDim $con($beg,$end) $max2}] == 1 } {
      gg::conRedimBegin
        gg::conRedim $con($beg,$end) $max2
      gg::conRedimEnd
  }
  if [catch {gg::conGetPt $con($beg,$end) -arc 0}] {
    set con($beg,$end) \
         [getConnectorByEndPoints $pt0 $pt1]
  }
 }
 ......
}
```

#3: H Domain Locator

- 3-D scaler implementation
  - (S1, S2, S3)
  - (S1) Length between index 1 and index 2 is approximately the scaled length in J direction.
  - (S2) Length between index 3 and index 4 is approximately the scaled length in K direction.
- Pinpoint the 4 H domain corners.
- Create butterfly connectors.
- Assemble H and O domains.
- Shape information check

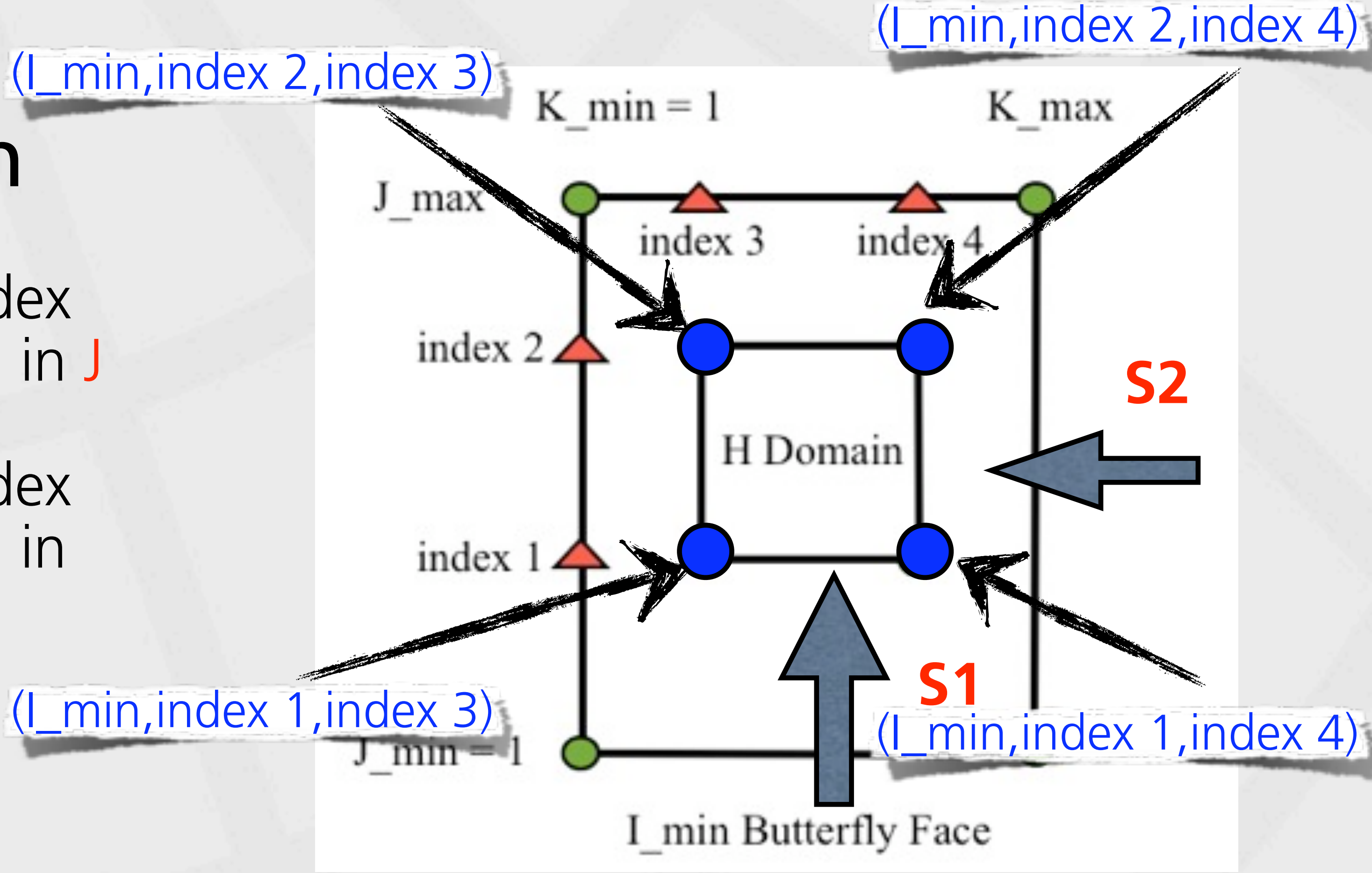


```
gg::dbImport $butterflyDBFile -style PLOT3D \
                    -format ASCII -precision DOUBLE

gg::domProject $butterflyDomList -type CLOSEST_PT \
                      -maintain_linkage
```

```
foreach end {ind3_min ind3_max} {
  foreach beg {ind1_min ind1_max} {
    set pt0 [gg::conGetPt $con($beg,$end) -arc 0]
    set pt1 [gg::conGetPt $con($beg,$end) -arc 1]
    if { [catch {gg::conDim $con($beg,$end) $max2}] == 1 } {
        gg::conRedimBegin
          gg::conRedim $con($beg,$end) $max2
        gg::conRedimEnd
    }
    if [catch {gg::conGetPt $con($beg,$end) -arc 0}] {
      set con($beg,$end) \
            [getConnectorByEndPoints $pt0 $pt1]
    }
  }
  ......
}
```

#3: H Domain Locator

- 3-D scaler implementation
  - (S1, S2, S3)
  - (S1) Length between index 1 and index 2 is approximately the scaled length in J direction.
  - (S2) Length between index 3 and index 4 is approximately the scaled length in K direction.
- Pinpoint the 4 H domain corners.
- Create butterfly connectors.
- Assemble H and O domains.
- Shape information check

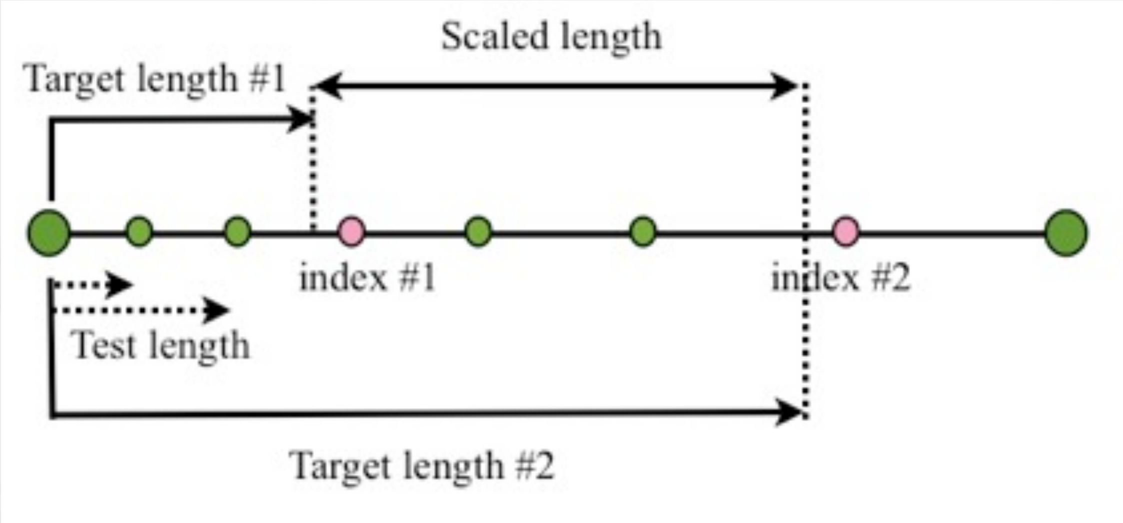


```
gg::dbImport $butterflyDBFile -style PLOT3D \
                    -format ASCII -precision DOUBLE

gg::domProject $butterflyDomList -type CLOSEST_PT \
                    -maintain_linkage
```

```
foreach end {ind3_min ind3_max} {
  foreach beg {ind1_min ind1_max} {
   set pt0 [gg::conGetPt $con($beg,$end) -arc 0]
   set pt1 [gg::conGetPt $con($beg,$end) -arc 1]
   if { [catch {gg::conDim $con($beg,$end) $max2}] == 1 } {
       gg::conRedimBegin
         gg::conRedim $con($beg,$end) $max2
       gg::conRedimEnd
    }
   if [catch {gg::conGetPt $con($beg,$end) -arc 0}] {
     set con($beg,$end) \
           [getConnectorByEndPoints $pt0 $pt1]
    }
  }
  ......
}
```

#4: Point Snapping Method

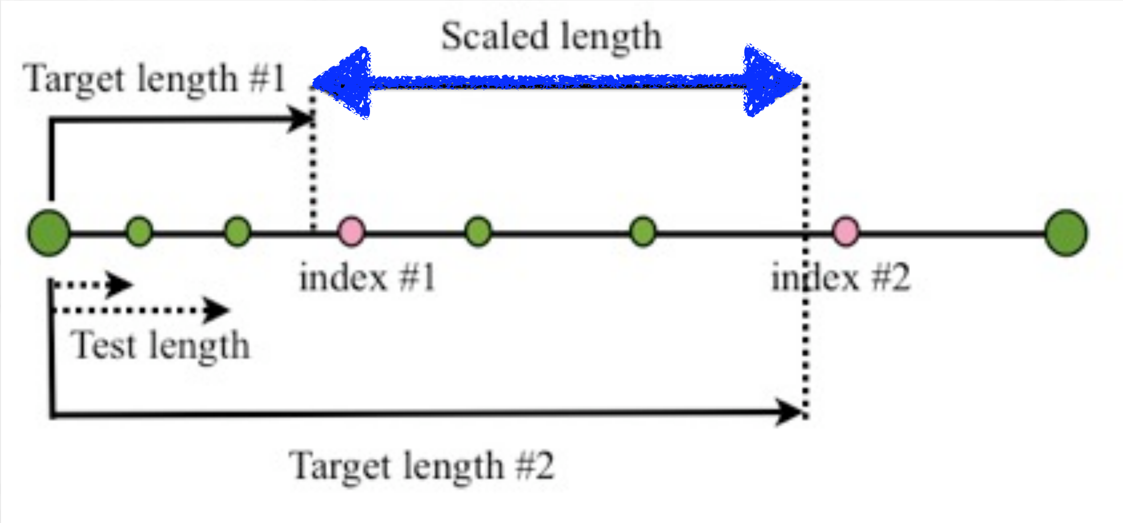
- Snap a point when one of the following conditions is met:
  - The test length is close to the target length #1.
  - The test length is close to the target length #2.
  - The difference between the test and target length #1 is smaller than the local spacing.
  - The difference between the test and target length #2 is smaller than the local spacing.
- Snapped point checkup
  - No point/1 point/2 points or more than 2 points are snapped.
  - Different points are snapped at block interfaces in the propagating direction.



```
set iL 0.0
set corner1_Pts {}

set targetL_1 [expr $iL *(1.0-[lindex $oScaleFac 0]) / 2.0]
set targetL_2 [expr $iL - $targetL_1 ]
set testL 0.0

for { set ii 1 } { $ii < $max1 } { incr ii 1 } {
  set testL [expr $testL + [lindex $iSpacing [expr $ii-1]]]
  if { [expr abs( $targetL_1 - $testL )] < $tol || \
        [expr abs( $targetL_2 - $testL )] < $tol || \
        [expr abs( $testL - $targetL_1)] < \
        [lindex $iSpacing [expr $ii-1]] || \
        [expr abs( $testL - $targetL_2)] < \
        [lindex $iSpacing [expr $ii-1]] } {
      lappend corner1_Pts [expr $ii+1]
   }
}
set ogrid_i [lindex $corner1_Pts 0]
```

#4: Point Snapping Method

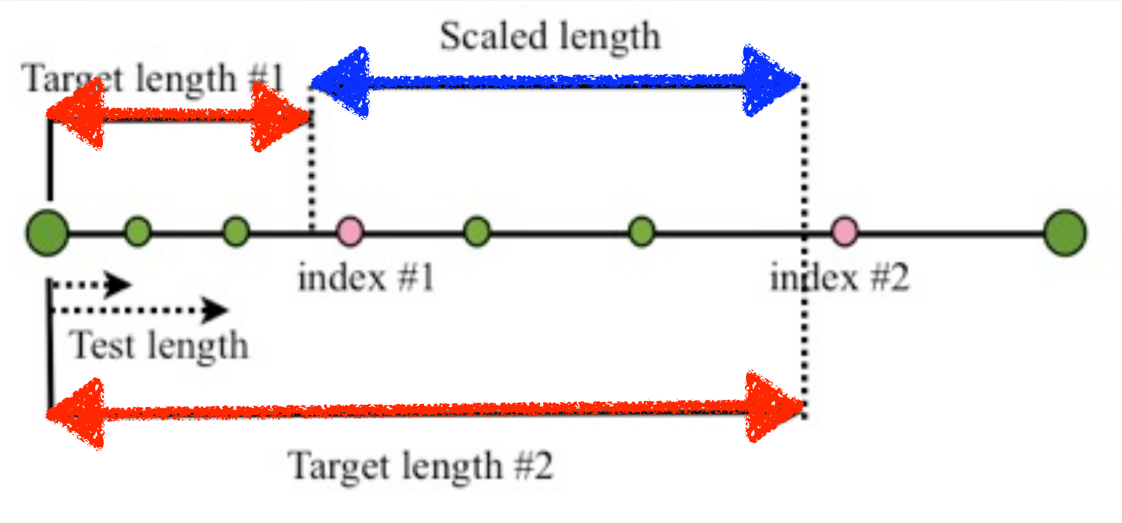
- Snap a point when one of the following conditions is met:
  - The test length is close to the target length #1.
  - The test length is close to the target length #2.
  - The difference between the test and target length #1 is smaller than the local spacing.
  - The difference between the test and target length #2 is smaller than the local spacing.
- Snapped point checkup
  - No point/1 point/2 points or more than 2 points are snapped.
  - Different points are snapped at block interfaces in the propagating direction.



```
set iL 0.0
set corner1_Pts {}

set targetL_1 [expr $iL *(1.0-[lindex $oScaleFac 0]) / 2.0]
set targetL_2 [expr $iL - $targetL_1 ]
set testL 0.0

for { set ii 1 } { $ii < $max1 } { incr ii 1 } {
  set testL [expr $testL + [lindex $iSpacing [expr $ii-1]]]
  if { [expr abs( $targetL_1 - $testL )] < $tol || \
       [expr abs( $targetL_2 - $testL )] < $tol || \
       [expr abs( $testL - $targetL_1)] < \
       [lindex $iSpacing [expr $ii-1]] || \
       [expr abs( $testL - $targetL_2)] < \
       [lindex $iSpacing [expr $ii-1]] } {
     lappend corner1_Pts [expr $ii+1]
  }
}
set ogrid_i [lindex $corner1_Pts 0]
```

#4: Point Snapping Method

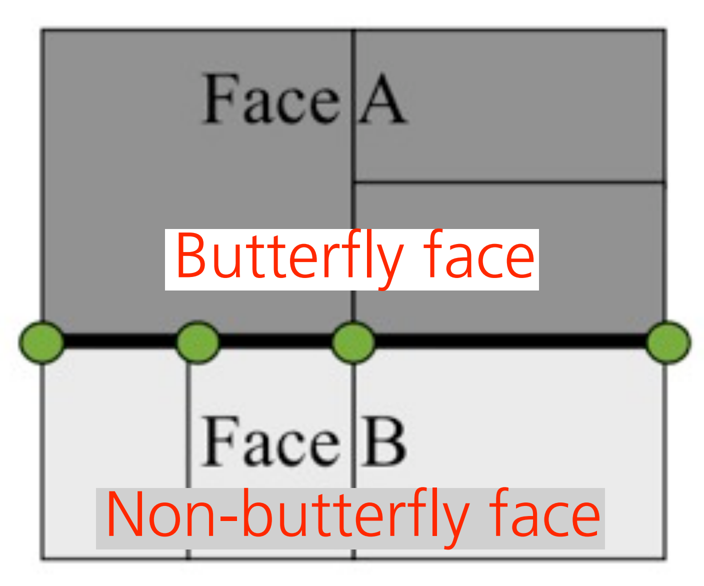
- Snap a point when one of the following conditions is met:
  - The test length is close to the target length #1.
  - The test length is close to the target length #2.
  - The difference between the test and target length #1 is smaller than the local spacing.
  - The difference between the test and target length #2 is smaller than the local spacing.
- Snapped point checkup
  - No point/1 point/2 points or more than 2 points are snapped.
  - Different points are snapped at block interfaces in the propagating direction.



```
set iL 0.0
set corner1_Pts {}

set targetL_1 [expr $iL *(1.0-[lindex $oScaleFac 0]) / 2.0]
set targetL_2 [expr $iL - $targetL_1 ]
set testL 0.0

for { set ii 1 } { $ii < $max1 } { incr ii 1 } {
  set testL [expr $testL + [lindex $iSpacing [expr $ii-1]]]
  if { [expr abs( $targetL_1 - $testL )] < $tol || \
        [expr abs( $targetL_2 - $testL )] < $tol || \
        [expr abs( $testL - $targetL_1)] < \
        [lindex $iSpacing [expr $ii-1]] || \
        [expr abs( $testL - $targetL_2)] < \
        [lindex $iSpacing [expr $ii-1]] } {
      lappend corner1_Pts [expr $ii+1]
   }
}
set ogrid_i [lindex $corner1_Pts 0]
```

#5. Multi-domain Butterfly Face

- Key tasks:



  - Detect shared edge.
  - Sort cons on a multi-con edge.
  - Remove original cons that are inside of butterfly faces.

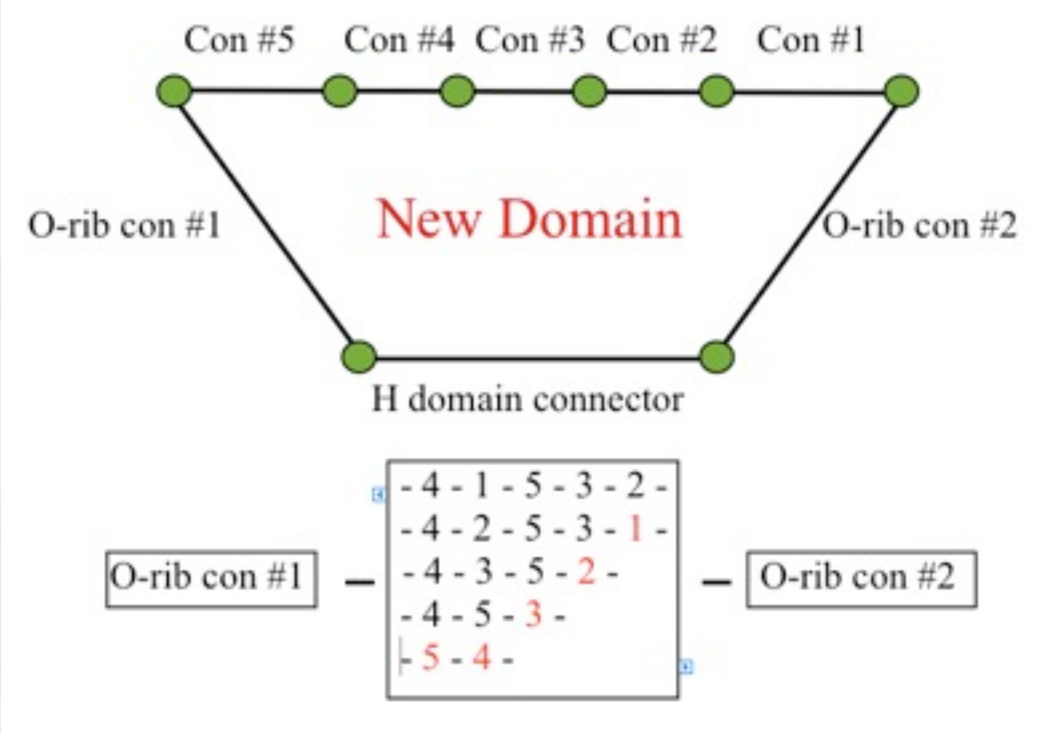
- Strategy evaluation:
  - ⊗ Join butterfly domains into single H-domain.
  - ☺ Keep butterfly domains and locate new cons using the global indices of butterfly block face (not domains).

```
proc GetBlkEdgeCons {blk face1 face2} {
  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 } {
                  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
              }
          }
      }
  }
  ......
  return $sharingCons
}
```

```
gg::domJoinBegin $dom_1
  gg::domJoinAddDom $domList
gg::domJoinEnd
```

#6. Shared Edge Manager

- Two edge structures:
  - Single-con edge: O-rib con, H cons (multi-segment).
  - Multi-con edge: other.
- Multi-con edge manager is used for sorting connectors on an edge.

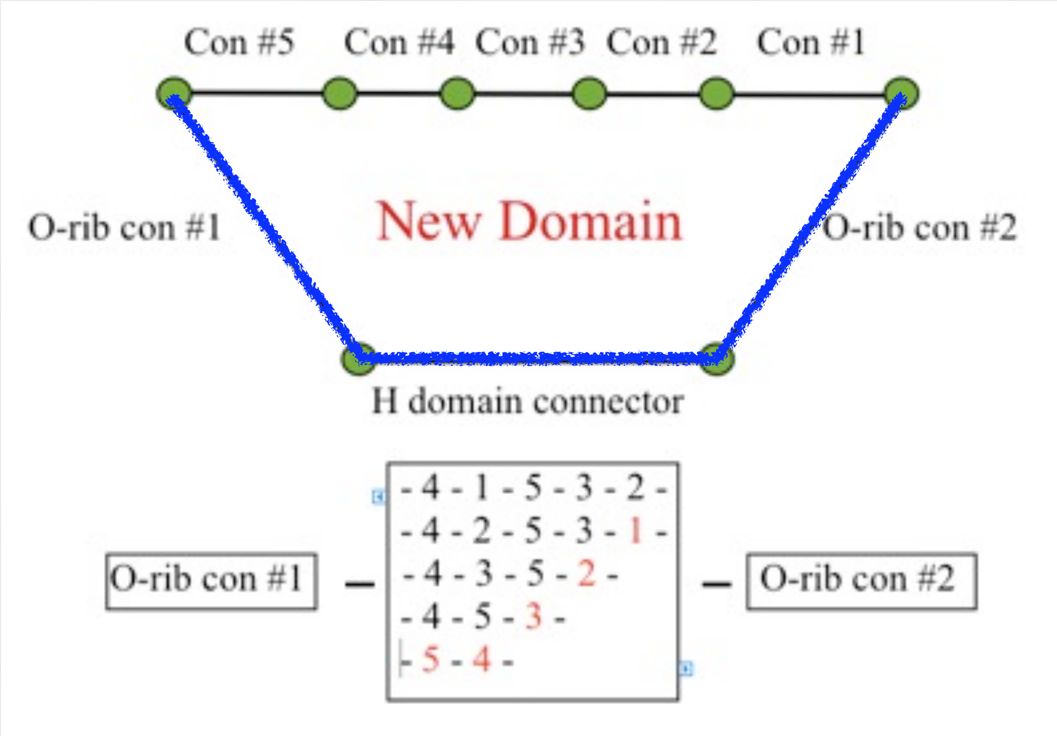


- Initial order: 4 - 1 - 5 - 3 - 2.
- Target order: 5 - 4 - 3 - 2 - 1.
- Iteration numbers: 4

```
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
}
```

#6. Shared Edge Manager

- Two edge structures:
  - Single-con edge: O-rib con, H cons (multi-segment).
  - Multi-con edge: other.
- Multi-con edge manager is used for sorting connectors on an edge.

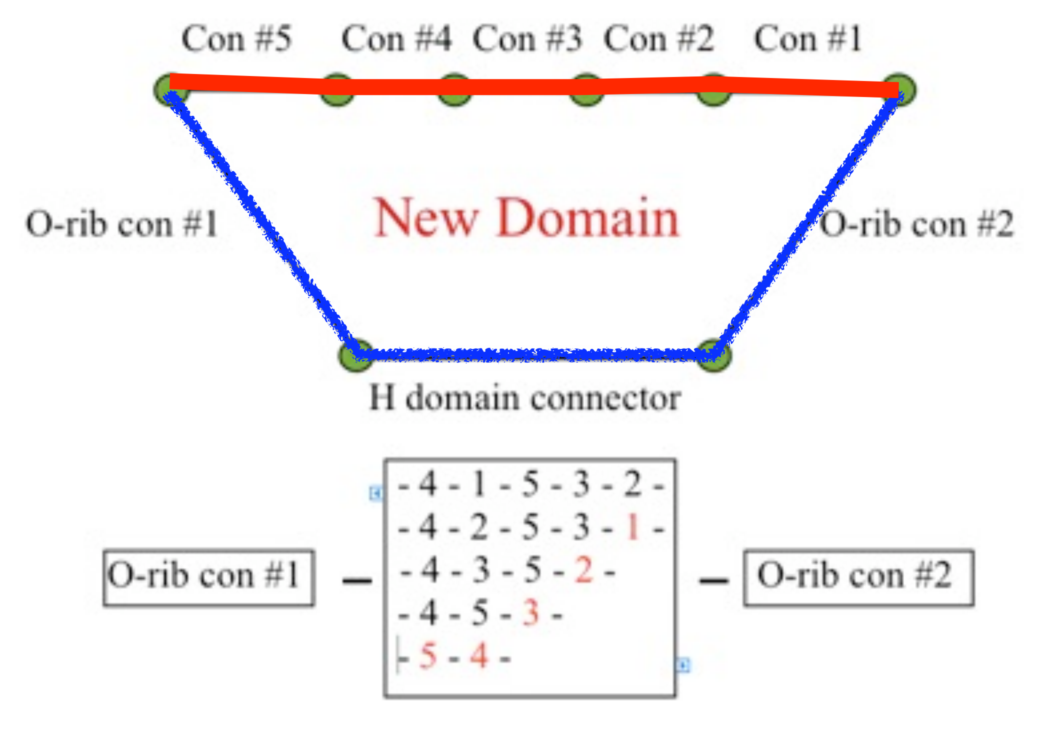


- Initial order: 4 - 1 - 5 - 3 - 2.
- Target order: 5 - 4 - 3 - 2 - 1.
- Iteration numbers: 4

```
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
}
```

#6. Shared Edge Manager

- Two edge structures:
  - Single-con edge: O-rib con, H cons (multi-segment).
  - Multi-con edge: other.
- Multi-con edge manager is used for sorting connectors on an edge.



- 4 - 1 - 5 - 3 - 2 -
- 4 - 2 - 5 - 3 - 1 -
- 4 - 3 - 5 - 2 -
- 4 - 5 - 3 -
- 5 - 4 -

- Initial order: 4 - 1 - 5 - 3 - 2.
- Target order: 5 - 4 - 3 - 2 - 1.
- Iteration numbers: 4

```
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
}
```

Future Work

- Rewrite the script for Pointwise using Glyph 2.0.
- Handle other grid topologies: C-H, L-H and O-grid around bodies.
- Add custom libraries for specific distributions of O-rib connectors.
- Allow arbitrary H-domain (non-center) locations to be defined by users.
- Optimize frequently used components to boost the script performance.
- Allow user to examine grid before it is saved.
- Improve the user interface using advanced Tk widgets.
- Explore the possibility of parallelizing the script for large applications.

POINTWISE®
Reliable CFD Meshing