Advanced Grid Generation for Engineers and Scientists

Griddle 2.0
User Manual

Contents

Introduction

This document describes the use of *Griddle* grid generation tools in conjunction with the *Rhino* CAD system.

Griddle offers engineers and scientists both automatic and interactive grid generation capabilities that cover a wide range of volume grid generation needs as well as operations with surface meshes. *Griddle* provides new tools as well extensions to meshing tools available within *Rhino* CAD system.

Detailed tutorial examples of geomechanical applications are provided in *Griddle 2.0 Tutorial Examples* document that can be accessed via *Windows Start Menu → Itasca Griddle 2.0.* The files for the tutorial examples can be accessed by clicking on *Griddle 2.0 Tutorial Files* link. The link opens user writable directory `ProgramData\Itasca\Griddle200` (typically on drive "C:") which contains the documentation and the tutorial material. Users can directly work in this directory. A reserve copy of the same material can be found in the `Documentation` subfolder of *Griddle* installation location (typically in `C:\Program Files\Itasca\Griddle200`).

Quick Start for *Griddle 2.0* and *Rhino*

Griddle 2.0 is a suite of tools for surface- and volume-meshing within the *Rhino* CAD system. **Griddle 2.0 is installed as a *Rhino* plugin and is only available for the Windows version of *Rhino 6* and *Rhino* 7.**

The *Griddle 2.0* installer automatically removes all previous versions of *Griddle* installed on the machine, including those for *Rhino 5* and *Rhino 6*.

Rhino 6 and/or *Rhino* 7 are installed separately from *Griddle* and are used for constructing model geometry.

First Time Use After Installation

- If previous versions of *Griddle* have never been installed, nothing else needs to be done and *Griddle 2.0* can be readily used. *Griddle 2.0* automatically integrates with *Rhino 6* and *Rhino 7;* the *Griddle* toolbar should be visible within the *Rhino* workspace:

Figure 1: The *Griddle 2.0* toolbar.

- If *Griddle 1.0* was previously installed, additional steps may be needed to remove parts of the older *Griddle* from *Rhino*[1]. After this one-time clean-up, *Griddle 2.0* is ready to be used. The steps below are outlined for *Rhino 6* but they can also be applied to other versions of *Rhino* to fully remove *Griddle 1.0* components.

  1. Open *Rhino*. There may be two *Griddle* toolbars: an old one from v1.0 and a new one from v2.0.
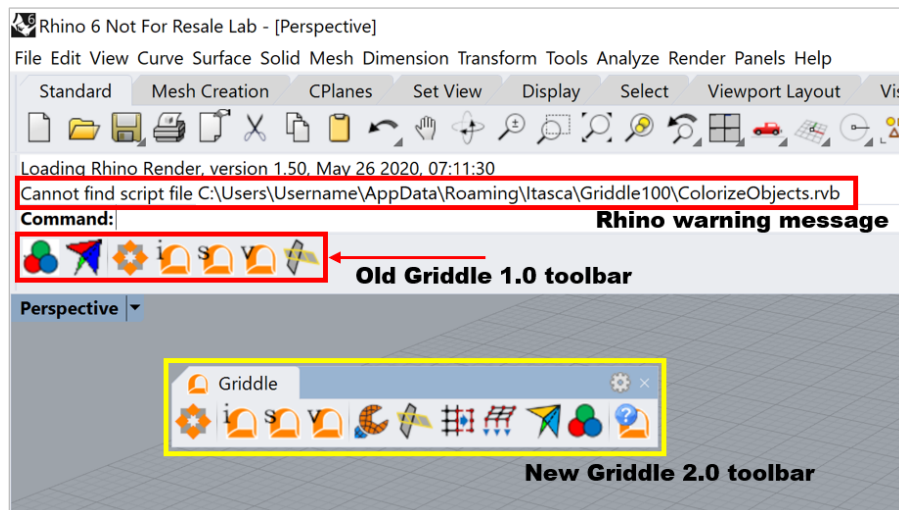
Figure 2: The *Rhino 6* window with *Griddle 1.0 and Griddle 2.0* toolbars.

---

[1] *Griddle 1.0* used *Rhino* integration tools and required a multi-step installation process (separate installation of plugins for each command, toolbars, and scripts). *Griddle 2.0* operates differently – it installs all components at once. During installation *Griddle 2.0* attempts to remove its older versions from the system, but some parts may remain and should be removed manually.

2. The *Griddle 1.0* toolbar should be removed from *Rhino*. To do this, navigate to the top menu in *Rhino* and click on **Tools → Toolbar Layout…** This displays toolbars that are open/loaded in *Rhino*. Find "Griddle100", right click on it and select **Close** (do not **Save** when offered).

WARNING: DO NOT CLOSE "Griddle" TOOLBAR - it is used by *Griddle 2.0*
ONLY CLOSE "Griddle100" TOOLBAR
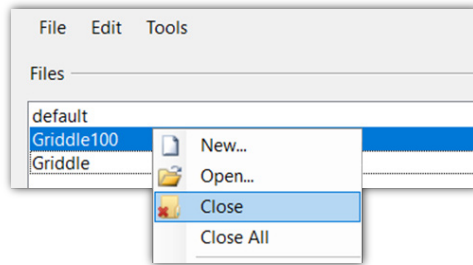
Figure 3: Closing *Griddle 1.0* toolbar from *Rhino*'s Toolbar Layout menu.

3. Figure 2 shows that *Rhino* outputs a warning "Cannot find script file …ColorizeObjects.rvb". This message may appear when *Rhino* starts. This occurs because the *Griddle 2.0* installer removes *Griddle 1.0* script files, however, *Rhino* is not aware of that (as these files were originally installed manually). Ignore this warning or remove the reference to old scripts from *Rhino* to prevent future warnings. To do so, use the menu items **Tools → Options → RhinoScript**. Find "ColorizeObjects.rvb" in the list of scripts and delete it (click on the large red X).

Griddle 2.0 Components

This section provides a brief summary of *Griddle 2.0* components. Each component can be accessed by clicking on an icon on *Griddle* toolbar or by typing a command that is the same as the component name (if *Rhino* is set up to use language other than English, type an underscore before the command, e.g. **_GInt**).

Table 1. *Griddle* components.

Icon	Component name, Command (English)	Description
	BlockRanger	Structured hexahedral volume mesher that operates on solids
	GInt	Surface mesh intersector for making meshes conformal
	GSurf	Unstructured surface remesher to create unstructured surface meshes with specified parameters
	GVol	Unstructured tetrahedral/hex-dominant volume mesher
	GHeal	A set of tools to identify and fix surface mesh problems
	GExtract	A set of tools to extract pieces of surface meshes based on user-specified criteria
	GExtend	A set of tools to expand surface meshes along a specified boundary
	GExtrude	A set of tools to extrude surface mesh along its boundary to create a watertight domain
	NonManifoldMerge	*Rhino* command to merge surface meshes and create a single non-manifold mesh
	ColorizeObjects	Assign random colors to objects
	GriddleAbout	Display information about *Griddle* and check for updates

Detailed descriptions of these components are provided in later sections of this manual.

Griddle 2.0 License Key

Griddle 2.0 requires a license key to run all components with full functionality. A license key can be obtained by contacting www.itascacg.com/sales.

BlockRanger is now integrated in *Griddle 2.0* and therefore it requires a *Griddle 2.0* license. Starting with *Griddle 2.0*, **BlockRanger** will not accept licenses from *FLAC3D* or *3DEC*.

If a license key is not present, *Griddle 2.0* will operate in demonstration mode. Demonstration mode limits functionality as described in Table 2.

<p align="center">**Table 2.** *Griddle 2.0* **demonstration mode limits.**</p>

Icon	Component	Limitations in demonstration mode
	BlockRanger	Saves output volume mesh in VRML format only
	GInt	• Functionality to keep meshes separate (*OutputMesh=Separate*) is not available (all meshes will be merged in the output) • *SplitIntersections* option is not available
	GSurf	• Number of elements in the output mesh is limited to 5000 • Functionality to keep meshes separate (*OutputMesh=Separate*) is not available (all meshes will be merged in the output)
	GVol	Number of elements in the output mesh is limited to 10000
	GHeal	Automatic mesh repair, *AutomaticHeal*, is not available
	GExtract	Only separation of a single surface (*SingleSurface*) is available in demonstration mode
	GExtend	Not available in demonstration mode
	GExtrude	Not available in demonstration mode
	NonManifoldMerge	Fully available in demonstration mode
	ColorizeObjects	Fully available in demonstration mode
	GriddleAbout	Fully available in demonstration mode

If a network key is purchased, follow these steps:

1. <u>Do not</u> connect the network-version hardware key to a server. First install *Sentinel Keys Server* drivers on the server machine. *Sentinel Keys Server* is included in "*Sentinel Protection Installer 7.6.9*" (or any newer version) and it can be downloaded from: https://www.itascacg.com/software/support/utilities or https://sentineltechsupport.gemalto.com/2017/10/sentinel-protection-installer-7-6-9-released/
2. Connect the *Griddle* network-version hardware key to the server computer.
3. *Griddle* should be installed on a network accessible user workstation (not on the server).

To set up access to the server network key from the workstation, start *Rhino* on the workstation, and click on any *Griddle* icon which would require the license (any of the first eight components in Table 2). A dialog will appear prompting user to specify the server location (the machine with the *Griddle* key). Click on "*Use Network Key*" and specify the server IP address (the default value of "*localhost*" corresponds to the user workstation, not the server).

Updating *Griddle 2.0*

Griddle 2.0 is continuously being improved. Users may expect periodic updates. *Griddle* automatically checks for available updates each time *Rhino* is closed. If updated version is available, a message box will appear offering users to navigate to the download page.

Users may also check for updates manually by clicking on the icon on *Griddle* toolbar or typing the **_GriddleAbout** command. Besides displaying technical information about *Griddle* and license information, the **Griddle About** dialog notifies users when updates are available.

Users may also check for updates by navigating to the *Griddle* update page:
http://www.itascacg.com/ftp_pub/griddle/v200/griddle200.html

The installer for *Griddle* updates removes previous versions prior to installing the update.

Brief Introduction to the *Rhino 6 and 7* Workspace

The *Rhino* Command Area, Command Prompt, and Commands

All *Rhino* operations are performed by commands. The command area and command prompt are shown in Figure 4.

The command area is the field (generally on top) where *Rhino* displays system and command output information. Below the command area is the command prompt, where users can type *Rhino* or *Griddle* commands. The location of the command area can be moved by dragging.

When a user clicks on any icon (including icons from the *Griddle* toolbar) or a menu item, a corresponding command is automatically issued and placed into the command prompt. For example, clicking on ![icon] from the *Griddle* toolbar issues the command **_GSurf,** as shown in Figure 4.
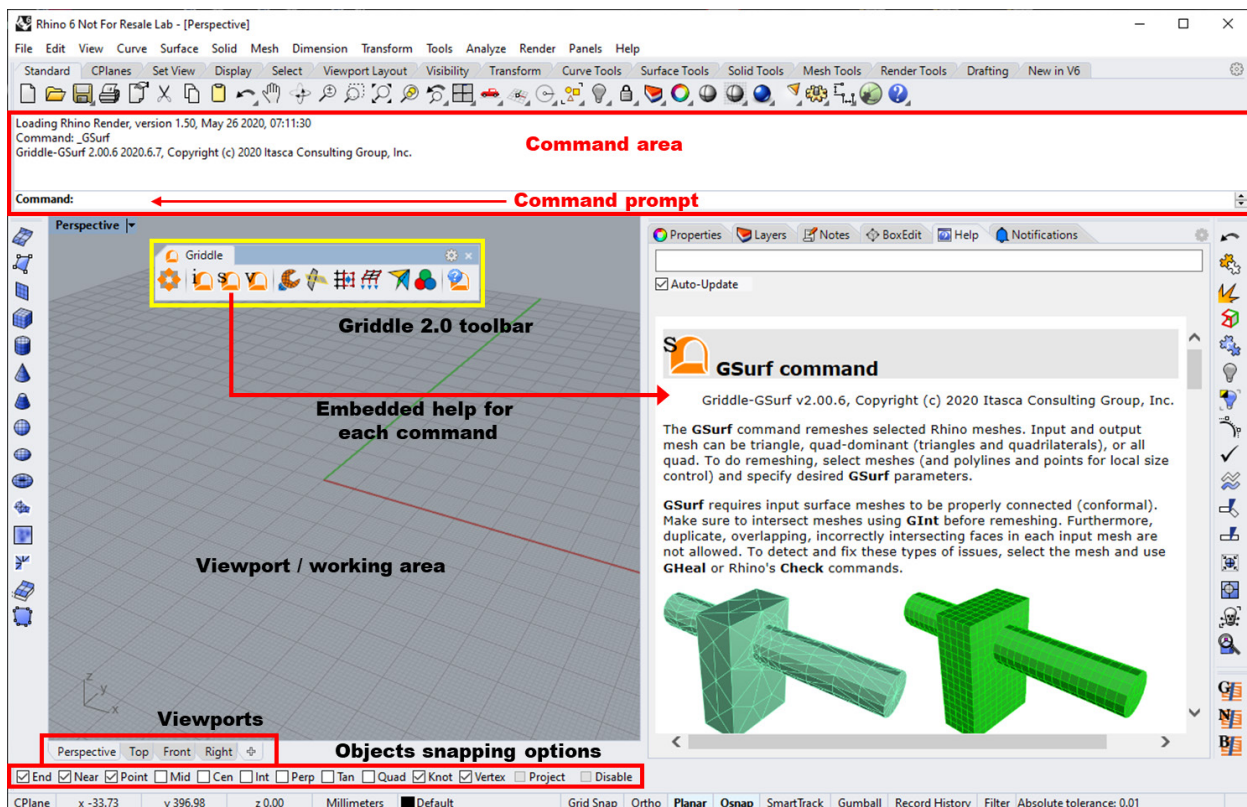
Figure 4: The *Rhino* window featuring the *Griddle* toolbar and the *Help* pane.

Another example is the command **_CommandHelp,** which opens the *Help* pane. If the Auto-Update checkbox is selected in the *Help* pane, it will display information about currently typed command, including *Griddle* commands, as shown in Figure 4. The same *Help* pane can be opened using **Help →** **Command Help** on the menu. The *Help* pane can be dragged (by grabbing the tab) and placed into any convenient location.

Note that if an underscore is placed before the command name (e.g. _**Shade**), *Rhino* assumes that the user refers to the English command name (i.e., **Shade**). Therefore, regardless of the language in which *Rhino* is set up, _**Shade** creates a shaded image whereas the command **Shade** will only be understood if the installation language is English.

Throughout this manual, *Griddle* and *Rhino* functions are mostly referred through commands that are entered or issued in the *Rhino* command prompt. Sometimes *Rhino* functions are referred to by their icon name or through a menu item. The name of an icon is displayed in a tooltip that pops up when user hovers the mouse over the icon. Often, *Rhino* icons have two related functions depending on whether the user clicks the left or right mouse button (e.g., the commands _**PointsOn** and _**PointsOff** corresponding to the *Rhino* icon).

Viewports, Rotating, and Panning Views

Viewports are the part of the *Rhino* workspace where the user manipulates models. There are four standard viewports: *Perspective*, *Top*, *Front*, and *Right*, as shown in Figure 4. In the *Perspective* viewport, which presents a model in 3D perspective view, the user can rotate the view (right-click and hold) or pan the view using **Shift** and the right mouse button held down. The other viewports provide 2D views that only allow panning. The user may display several viewports simultaneously and may even create custom viewports.

Object Snapping Options

Two types of snapping are available in *Rhino*: **Grid Snap** and **Osnap**. When **Grid Snap** is enabled, moving the mouse while it controls an object (e.g., dragging or extending an object) will cause the mouse pointer to snap to discrete positions in space corresponding to the X, Y, Z position of grid nodes.

To be able to snap lines, polylines, corners of objects, etc. to existing objects, **Osnap** must be active. Click on the **Osnap** at the bottom of the *Rhino* window to activate it. Next, specify which particular point(s) of an existing object may be snapped to by checking any of the words: **End**, **Near**, **Point**, **Mid**, etc. that appear at the bottom the *Rhino* window, as shown in Figure 4.

Both snapping options can be active simultaneously.

Units and Tolerances

Rhino allows users to specify units when creating a new project. Units determine what is displayed when measuring distance, location, or angles. More importantly, units and object sizes are related to tolerances used by *Rhino*. By default, tolerance is determined based on model units. However, users can specify custom values (see Figure 5).

Tolerances are very important when intersecting or doing Boolean and other operations with NURBS or SubD surfaces or BRep[2] objects or when applying *Rhino* commands to meshes (note that *Griddle*'s **GInt**

---

[2] NURBS – non-uniform rational basis spline, SubD surfaces are high-precision Catmull-Clark subdivision surfaces (only in *Rhino 7*), BRep – boundary representation;

tool has its own user-specified tolerance). Because these types of operations often do not provide exact (analytical) results but use approximate calculations or representations of objects, using proper tolerances ensures that operations are done as desired and in reasonable timespans.

In all cases, make sure that the model is never too far away from the origin and, if it is, **_Move** it closer to the origin. Moving objects closer to the origin improves accuracy of operations and helps with the graphics both in *Rhino* and in the analysis software.
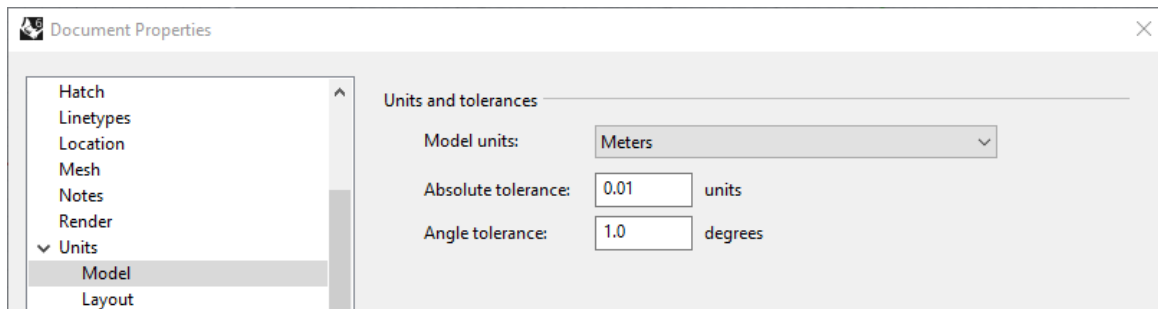
Figure 5: The *Rhino* window to specify model units and tolerances.

When starting a new project, use the appropriate *Rhino* template to specify whether the model will be using "Large Objects in Meters", "Small objects in feet", etc. After that create a model or import DXF, STL files, or even other existing 3dm files into the new project. This technique controls the tolerance (instead of using an inherited tolerance when starting from an imported file).

More information on tolerance can be found in the References section (see: Understanding *Rhino* tolerances).

Orthogonal Restriction of Mouse Movement

Mouse movement can be restricted permanently to the X, Y and Z directions by clicking the word **Ortho** that appears at the bottom of the graphic window — or by pressing **F8**. Mouse movement may be temporarily restricted by holding down **Shift** while moving the mouse.

Thicker lines

The default representation of curves (1 pixel wide) may be too thin and difficult to see in a working area or when a *Rhino* window is projected on a screen. Figure 6 shows, at left, the default representation. The following procedure shows how to draw curves 3 pixels-wide whenever **Shaded View** is selected (Figure 6, right).

Within *Rhino*, Select the **Tools → Options** menus item. In the left pane of the **Rhino Options** dialog, select **View → Display Modes → Shaded → Objects → Curves**. In the *Curve Settings* pane, to the right, change the default setting of *Curve width* from 1 to 3 pixels and click **OK** (Figure 7). These changes are saved with the project.

NURBS surfaces – surfaces represented by non-uniform rational basis spline; SubD surfaces (only in *Rhino 7*) represented by generalized higher order splines for extra smoothness (Catmull–Clark subdivision surface); BRep objects – objects represented by its boundaries. All these objects have (semi-)analytical description.
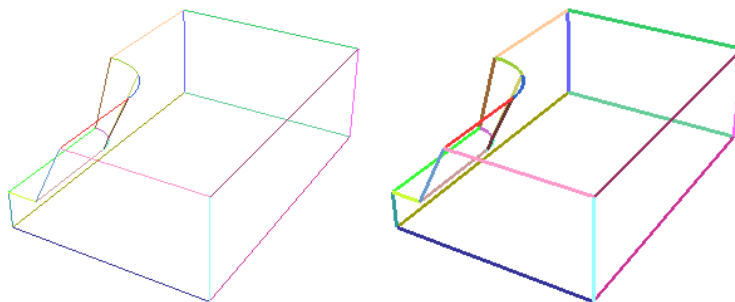
Figure 6: A regular (left) and "thick line" (right) representation.
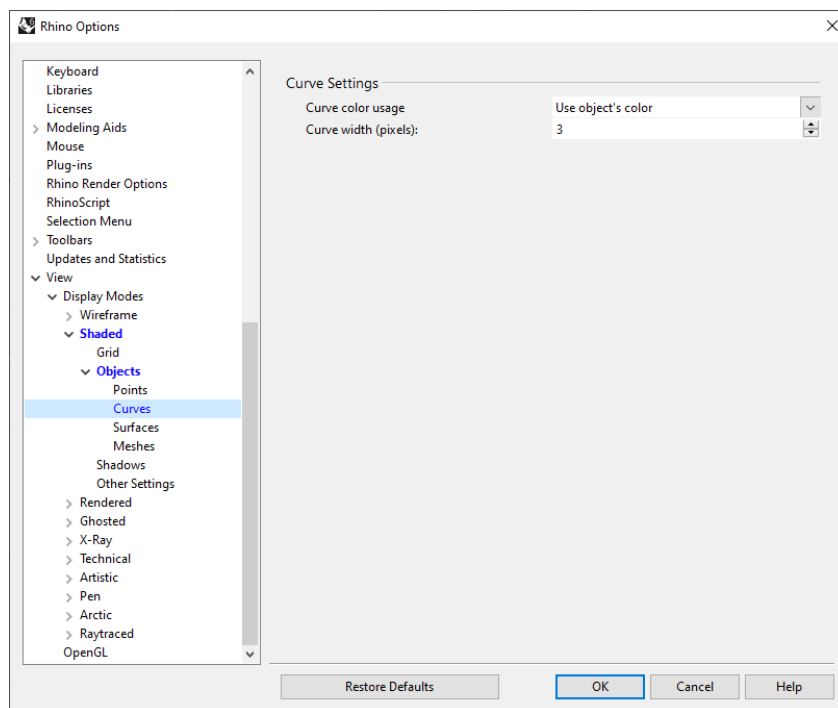
Figure 7: Setting line width to 3 pixels in Shaded View.

Hiding the Background Grid

Hide or show the background grid in an active viewport by pressing **F7**. To hide the grid in all viewports, go to **Tools → Options**. In the left pane of the **Rhino Options** dialog, click on **Document Properties → Grid**, and in the Grid properties section, uncheck *Show grid lines* and *Show grid axes*.

Avoiding Accidental Object Dragging

It is easy to inadvertently drag a highlighted object with the left mouse button. To avoid this, open menu item **Tools → Options**. In the left pane of the **Rhino Options** dialog, click on **Rhino Options → Mouse**, and in the *Click and drag* section, set the *Object drag threshold* to 100 pixels. Now, dragging an object requires a minimum of 100 pixels of mouse movement before it takes effect.

A number of other customizations and helpful tips/information about *Rhino* can be found in the *Rhino* documentation (press **F1** or navigate to **Help** menu).

Structured and Unstructured Meshes

Griddle creates structured volume meshes using **BlockRanger** and unstructured volume meshes using **GVol**. **BlockRanger** operates on solids[3] represented by BRep or NURBS objects. **BlockRanger** <u>does not</u> operate on closed SubD objects. **GVol** operates on sets of conformal surface meshes (structured or unstructured) which compose watertight (closed volume) domains. After executing **BlockRanger** or **GVol**, output file(s) with volume mesh data are created. The output contains information about nodes, faces, elements, node-face connectivity, node-element connectivity, and element and surface groups[4]. A mesh from such output can be imported into numerical modeling software (e.g. *FLAC3D*, *3DEC*, *ABAQUS*, etc.)

<u>Structured meshes are identified by regular connectivity</u> and typically have well-shaped elements (zones). Simple examples of structured meshes are (see examples in Figure 8, Figure 10, and Figure 12):
- a quadrilateral mesh in 2D where each internal node is joined to 4 neighboring quadrilaterals, forming a regular array of elements, and
- a structured 3D hexahedral grid that has each internal node connected to 8 elements.

<u>An unstructured mesh is identified by irregular connectivity</u>. Surface unstructured meshes typically employ triangles and quadrilaterals, while an unstructured 3D volume mesh may contain tetrahedra, pyramids, prisms, hexahedra, and even more complex elements. *Griddle*'s components **GSurf** and **GVol** build triangular, quad-dominant, and pure quadrilateral surface meshes and tetrahedral and hex-dominant unstructured volume meshes, correspondingly.

Using an unstructured mesher typically provides the advantage of generating meshes for geometry of any complexity. It is typically much faster than the operations required to build a similar model with a mapped structured mesher (for cases when it is possible to use a structured mesher).

Two examples below (Figure 8, Figure 9) illustrate the difference between structured and unstructured volume meshes created for the same initial geometry. The model includes a layered domain with several parallel curving tunnels that are intersected by another tunnel.

Figure 8 shows a fully structured volume mesh that was created by decomposing the initial geometry into hexahedral, tetrahedral, and prism-like solids within *Rhino*. After such decomposition, the solids were meshed with **BlockRanger**. Preparing solids from the initial geometry in *Rhino* takes a considerable amount of time and effort. However, the resulting mesh is a high-quality, all-hexahedral structured mesh.

Figure 9 illustrates a fully unstructured volume mesh that was generated using **GVol**. The volume mesh was created from surface meshes of tunnels and rock layers. Manual decomposition into simple primitive shapes/solids is not required in this case. Overall model creation is much faster. The resulting mesh is a good-quality hexahedral-dominant unstructured mesh.

---

[3] Solids are assemblies of surfaces, called polysurfaces in *Rhino*, that have a clearly defined interior and exterior.
[4] Element and surface groups currently are available only for output in *FLAC3D*, *3DEC* and CSV formats.

For some models, like for the one presented in Figure 8 and Figure 9, it is also possible to create hybrid (mixed) meshes, in which the interior of the tunnels is structured mesh (meshed with **BlockRanger**) and the exterior volume is unstructured mesh (meshed with **GVol**). This is possible because tunnels have regular shape, while the overall shape of the model plus tunnel surfaces is irregular. These meshes would have to be created separately and output into separate files. Subsequently, they can be combined into a single mesh within numerical modeling software.
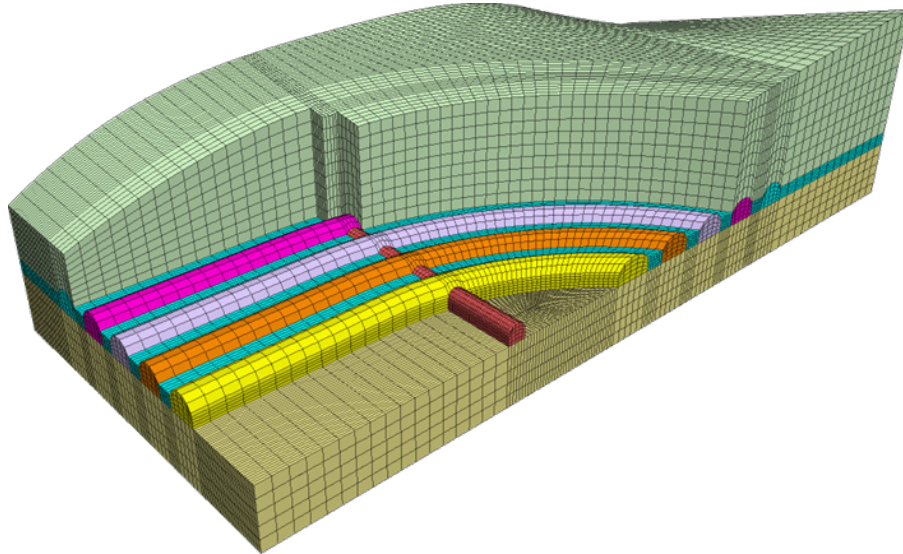
Figure 8: All hexahedral structured *FLAC3D* mesh generated by *BlockRanger*.
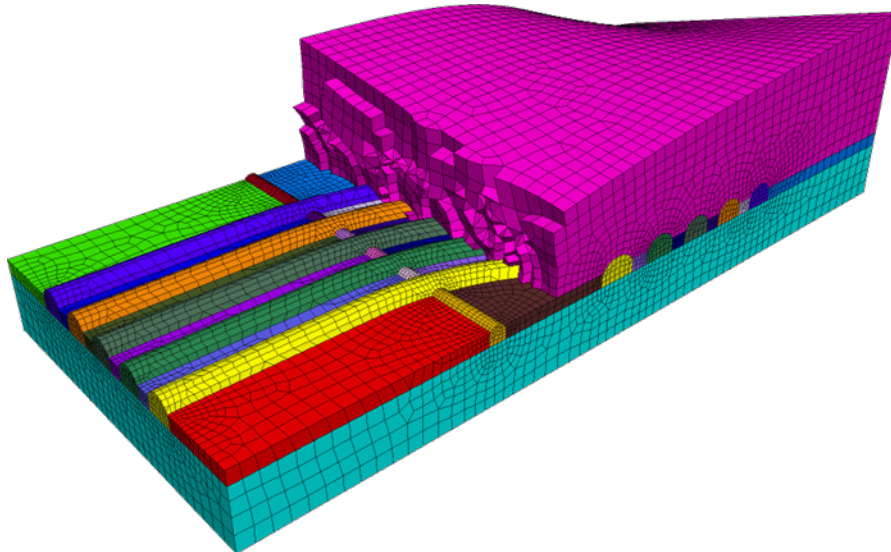
Figure 9: Hexahedral dominant unstructured *FLAC3D* mesh generated by *GVol*.

Using *Griddle* Tools for Structured Meshing - *BlockRanger*

BlockRanger is an interactive all-hex mapped mesher. The *Rhino* command to run ***BlockRanger*** is **_BR** and the icon to run it is provided in Table 1.

BlockRanger operates only on 4, 5 or 6-sided solids represented by a watertight BRep or NURBS objects (not SubD). It creates high-quality hexahedral (brick) volume mesh within such solids, which can be output in different formats for use in numerical modeling software.

Admissible ***BlockRanger*** solids (Figure 10) are:
- 6-sided solids (hexahedron-like) composed of 6 surfaces each bounded by 4 curves.
- 5-sided solids (prism-like) composed of 2 triangle-like surfaces bounded by 3 curves and 3 quad-like surfaces bounded by 4 curves.
- 4-sided solids (tetrahedron-like) composed of 4 triangle-like surfaces each bounded by 3 curves.
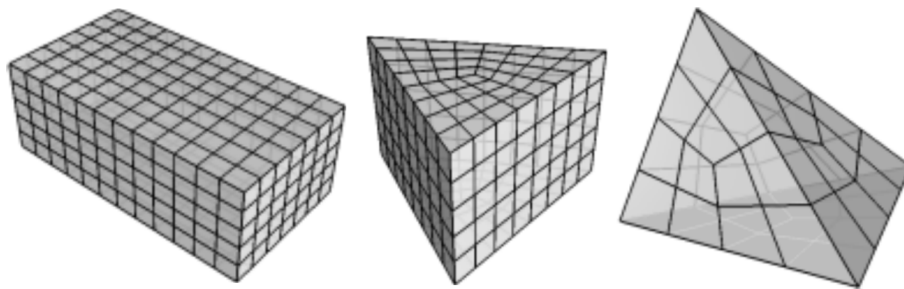
Figure 10: *Rhino* **solids recognized by** *BlockRanger* **with example** *BlockRanger* **mesh patterns for each.**

If the selected solids are contiguous, ***BlockRanger*** ensures that the resulting grid maintains grid conformity and continuity across block corners, edges, and faces so that no dangling nodes will result.

All solids successfully processed by ***BlockRanger*** are saved as grid files in a user-specified location or, if *AutoOutputName* option is specified – in the same location as the *Rhino* project. Solids that did not qualify or could not be successfully meshed remain highlighted onscreen. Use the **_Invert** and **_Hide** commands make these solids visible and correct them.

BlockRanger Options

BlockRanger options consist of two categories: Meshing settings and Output parameters. They are outlined below. If options have predefined or default values, they are provided in square braces.

Meshing Settings

MaxEdgeLength [*value ≥ 0, default = 0*]
Maximum element (zone) edge length in model coordinates. If a default value of 0 is specified, this number is calculated as one tenth of the length of the longest edge in the model.

MinEdgeResolution [*value > 0, default = 3*]
Minimum number of elements across each *Rhino* solid edge. Its default value is 3.

TargetNumElements [*value > 0, default = 1*]

Use this option to reduce the maximum element aspect ratio in the grid. **BlockRanger** will initially build a grid based on the prescribed *MaxEdgeLength* and *MinEdgeResolution*. In a typical block-structured grid, as **BlockRanger** reduces the maximum aspect ratio, the number of elements (zones) increases. **BlockRanger** stops when the number of elements exceeds *TargetNumElements*.

Output Parameters

GenerateSurfaceMesh *= [No (default), Yes]*

This option specifies if surface meshes should be created on the boundaries of each solid. Such surface meshes are composed of the external (surface) faces of the generated volume mesh (which is saved to a file). The surface mesh is not saved to the output file; it is only generated for use within *Rhino* model. Generated surface meshes can be used to visually check the quality of volume meshes or to create an unstructured volume mesh outside of the solids (for example, generating structured mesh inside a tunnel and unstructured mesh outside of it to fill a domain; then both structured and unstructured meshes can be imported and combined within numerical modeling software).

Note that if two solids are in contact with each other, surface meshes are created on each side of the contact to correspond to each individual solid.

OutputFormat *= [FLAC3D, 3DEC_5x, 3DEC_7x, ABAQUS, ANSYS, NASTRAN, LS-DYNA, VRML, CSV]*

Format in which the resulting volume grid file should be saved.

3DEC_5x stands for *3DEC* v5.0 or v5.2, and *3DEC_7x* stands for *3DEC* v7.0 or later. When outputting to *3DEC* format, a rigid block is generated for every element. This output is equivalent to *BlockType* = *Rigid* in **GVol**. **BlockRanger** currently does not have capability to output to *3DEC* deformable blocks format.

When outputting to CSV format, several output files will be created: a file with general information, separate files with information about elements, faces, and nodes (a file with internal surfaces will not contain any information as **BlockRanger** does not operate on such objects).

If the license key is not present, **BlockRanger** only allows outputting generated meshes in VRML format (see Demo limits in Table 2).

FormatType *= [Binary, Text]*

This option appears only if user selects *OutputFormat* = *FLAC3D* or *3DEC_7x*. The option specifies the format of the output file: in binary or text (ASCII). Saving and loading (reading) files in binary format is faster compared to the text format but binary files are not human-readable. All other *OutputFormat* choices save in text format only.

AutoOutputName *[= N/A (default), UserDefinedName]*

This option specifies a string that will appear in the output filename along with other identifiers. For example, for a *Rhino* project named "RhinoProject.3dm", a *FLAC3D* output in binary format will be named: "RhinoProject_UserDefinedName_Binary.f3grid". If user specifies a string in this option, the

Save As pop-up dialog will not appear and the output file will be named automatically as described above. If *AutoOutputName* is not specified (= "*N/A*"), a **Save As** dialog will appear, asking the user to provide the output file name and location. Note that:

- The value for this option is not saved: each time a user clicks on this option, a new string must be typed. To erase an existing string in *AutoOutputName*, click on the option and press **Enter**; this will cause a **Save As** dialog appear when executing *BlockRanger*.
- When using this option, the output file will be saved in the same location as the *Rhino* project file. If a file with same name already exists in that location it will be overwritten.

This option allows running *BlockRanger* in automatic mode without user interaction (i.e., the need to type file name and press **OK / Cancel** in the **Save As** dialog). This is particularly useful when invoking the **_BR** command from *RhinoScript*, *PythonScript*, or other *Rhino* tools and plugins.

Local Edge Resolution Control

BlockRanger controls local mesh resolution by manually setting individual edge resolution. A short example is provided below. Hands-on information can also be found in the *Griddle 2.0 Tutorial Examples, Tutorial 4: Creating a Structured Mesh with BlockRanger*. Figure 11 shows a model representing a 3D slope. The model consists of 8 solids.
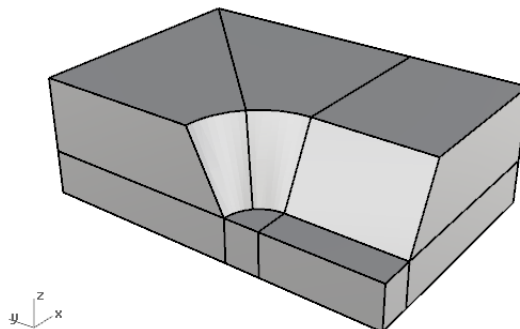
Figure 11: 3D slope example.

Running *BlockRanger* using the default parameters results in the grid as shown Figure 12.

Use the command **_DupEdge** to duplicate an edge from a *Rhino* solid and set the number of elements for that edge. For example, in Figure 13, the highlighted edge on the right side is duplicated first. Select duplicated edge and navigate to its *Properties* pane (or press **F3**). Enter number 8 in the Name field for the selected edge, which will be the resolution of the edge. A second edge is duplicated using **_DupEdge** (left side on Figure 13) and the number 5 is entered in its Name field.

If *BlockRanger* is run again, this time making sure to include the edges in the selection with the solids, the resulting grid will be more refined at the location of the duplicated edges (Figure 14). Set the option *GenerateSurfaceMesh=Yes* to visualize the resulting surface mesh in *Rhino*.
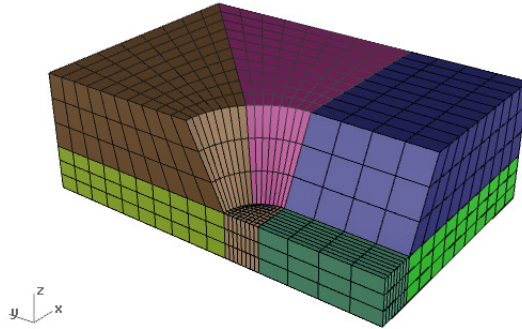
Figure 12: *BlockRanger* **generated mesh with default parameter values.**
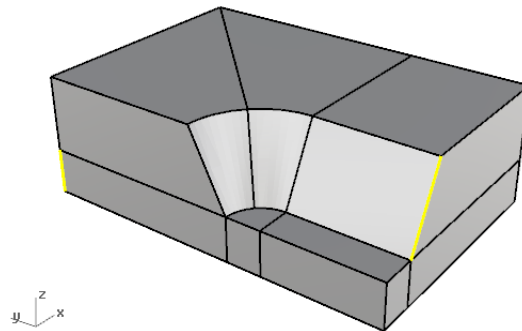
Figure 13: Duplicated edges (highlighted) with a custom number of elements specified.
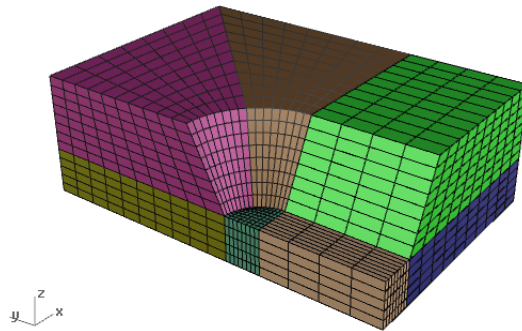
Figure 14: *BlockRanger* **generated mesh with specified edge resolution.**

Note that local edge resolution control takes precedence over automatically determined number of subdivisions per edge (based on the input parameters) only when it provides a larger number of elements for the edge (i.e., a finer mesh). Otherwise, the automatically determined number of elements is used.

Using *Griddle* Tools for Unstructured Meshing

Besides the structured volume mesher, **BlockRanger**, which operates on *Rhino* solids, *Griddle* includes tools to work with surface meshes and tools to generate fully conformal unstructured volume meshes. These tools are grouped within the *Griddle* toolbar (see Figure 1, Figure 15, and Table 1).

- **GInt** – surface mesh intersector (command: **_GInt**),
- **GSurf** – unstructured surface remesher (command **_GSurf**),
- **GVol** – unstructured tetrahedral/hex-dominant volume mesher (command: **_GVol**).

Figure 15: *Griddle* tools for unstructured meshing.

All three functions operate directly on *Rhino* surface meshes. The meshing workflow to generate an unstructured volume mesh is summarized in Figure 16.

| Start with a set of *Rhino* surface meshes (can be created by initial meshing of *Rhino* solids or by importing from *DXF*, *STL*, etc.) | → | Intersect all meshes using **GInt** to make them fully conformal | → | Remesh all meshes using **GSurf** to achieve desired mesh parameters (e.g. density, quality) while keeping meshes conformal | → | Create tetrahedral or hex-dominant volume mesh for watertight regions using **GVol**. Import volume meshes into numerical model. |

Figure 16: Typical workflow for unstructured meshing using *Griddle* tools.

Rhino has a rich set of surface meshing tools that allow users to create (e.g., by triangulation) and edit 3D surface meshes for objects represented by NURBS, SubD, or BRep surfaces and polysurfaces. These meshes, although good for machining and prototyping purposes, are usually not suitable for numerical computations. *Rhino*-generated (or imported) surface meshes often must be properly intersected and remeshed with **GInt** and **GSurf**, respectively. Once a desirable watertight set of surface meshes is obtained, it can be used (along with internal surface meshes) as an input to **GVol**, which fills the interior regions bounded by the surface meshes with tetrahedra or hex-dominant (hexahedra, prisms, pyramids and tetrahedral) elements for the use in numerical programs (such as *FLAC3D* or *3DEC*). The options for **GInt**, **GSurf**, and **GVol** are described below (and in the *Help* pane).

Numerous examples describing the use of **GInt**, **GSurf**, and **GVol** to create unstructured volume meshes are provided in *Griddle 2.0 Tutorial Examples*.

GInt – Surface Mesh Intersector

Griddle's tool **GInt** is a <u>surface</u> mesh intersector that can be applied to improperly intersected surface meshes to make them conformal.

- In conformal surface meshes, edges and nodes of mesh elements (faces) are fully shared between all elements connected to them.
- In conformal volume meshes, element faces, edges and nodes are fully shared between all elements connected to them.

Figure 17, left, shows a simple example of two non-conformal meshes in contact (blue and yellow meshes). After application of **GInt**, some of the initial faces are (randomly) split in triangular and quadrilateral faces to create conformal contact between meshes.
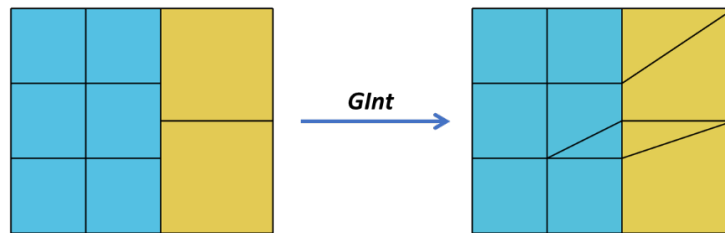
Figure 17: Example of non-conformal (left) and conformal (right) 2D meshes.

Figure 18 shows a more complex example of **GInt** operation on two meshes intersecting in non-conformal fashion (left image). After application of **GInt** to both meshes (right image), intersecting faces (darker green and pink) are split into numerous triangular faces to provide conformal contact.
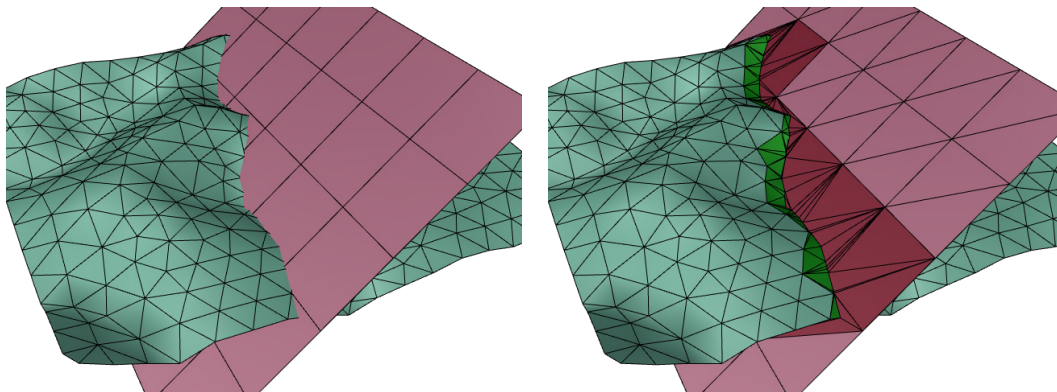
Figure 18: Example of non-conformal (left) and conformal intersecting surface meshes (right) after using *GInt*.

GInt can operate on multiple surface meshes at once to make the meshes themselves as well as intersections and contacts between them fully conformal. Use of **GInt** before continuing with meshing is typically needed as *Griddle* meshing tools **GSurf** and **GVol** require conformal surface meshes at input (see the workflow in Figure 16).

Note that **GInt** internally operates on triangular meshes only. Therefore, if an input surface mesh contains quadrilaterals, these quadrilaterals are automatically converted into triangles (by arbitrarily

inserting one of the two possible diagonal edges splitting the quad face) prior to **GInt** running (as in Figure 18). **GInt** has an option to restore some of the planar quadrilaterals (as in Figure 17) but they may not match the original ones.

If surface meshes are generated from *Rhino* NURBS, SubD, or BRep objects, it is better first to intersect such objects with *Rhino's* Boolean functions or merge them with the **_NonManifoldMerge** command. After that the objects should be meshed (triangulated) with the **_Mesh** command and then remeshed with **GSurf**. Intersecting/merging of NURBS, SubD, or BRep objects is more accurate than intersecting their mesh representation as meshes always approximate the initial analytical surfaces.

GInt Options

MeshType [= *Tri (default), Mixed*]
This option specifies the type of the output mesh. By default, it is a triangular mesh (*Tri*), however, if user selects the *Mixed* type, the output mesh will contain triangular and quadrilateral faces (elements). Quad faces are obtained by merging neighboring planar or near-planar triangular faces (only if the angle between face normals ≤ 0.5° and if the ratio of the diagonals in the resulting quad is less than 5). The *Mixed* option may be useful to preserve not-intersected quad faces in planar patches of the input meshes. In rare cases using this option may cause the output mesh to lose conformity (if two triangular faces across a previously non-conformal intersection are merged into a quad). This option is equivalent to using *Rhino*'s command **_QuadrangulateMesh**.

Tolerance [*value ≥ 0, default = 0*]
Tolerance is the most important parameter of **GInt**. It is an absolute distance (in model units) used to determine whether mesh faces intersect each other. If zero tolerance is specified, only those faces are intersected which are in exact contact with each other. Specifying non-zero tolerance allows searching for nearly intersecting faces (i.e., those within the tolerance). On the other hand, a very large tolerance may lead to a distorted output mesh as nodes within the tolerance are merged together.

Advanced Parameters

OutputMesh [= *Separated (default), Merged*]
By specifying *Separated* value, **GInt** will try to associate output mesh patches (sub-meshes) with the distinct input meshes and will place them into the layers they were originally obtained from. Patches that could not be associated with a source mesh are placed into a layer "MISMATCHED_PATCHES". When the value *Merged* is selected, a single output mesh will be generated and placed in the active layer.

SplitInersections [= *Yes, No (default)*]
If *Yes* is specified, **GInt** will split intersected faces from the rest of the mesh. If the user specifies *OutputMesh = Separated*, each set of split faces will be placed into a sub-layer "IntersectedFaces" of the original mesh layer. If the output mesh is merged, a new top-level layer named "IntersectedFaces" will be created, and all intersected faces will be placed in it (as a single merged mesh).

Note that if a high *Tolerance* is used, **GInt** may not only intersect faces between separate meshes but also may change faces within each mesh internally (mostly due to node merging). In such cases, if *OutputMesh* = *Separated*, the algorithm for detecting intersected faces may additionally report some non-intersected faces (therefore, *Tolerance* should be as small as possible when using this option).

Extracting intersected faces allows operation on them (e.g., change name/layer, delete layer, etc.) or assigning specific mesh sizes to them (for example, to densify meshes around the intersections).

DeleteInput [= *Yes (default), No*]
Specifying *Yes* indicates that the original selected meshes will be deleted. If *No* is selected, the original meshes will remain intact.

Reset resets advanced parameters to the default values.

After **GInt** completes the operation, it outputs a text (ASCII) log file with information about input and output meshes. The log file uses the name of *Rhino* project file (e.g. "RhinoProject.3dm") and adds "_GInt.log" to it (e.g. "RhinoProject_GInt.log"). The file is saved in the same directory where *Rhino* project is saved.

GSurf – Surface Remesher

Griddle's tool **GSurf** is a surface remesher that is used to remesh selected surface meshes to desired element size (on average) and type (triangle, quad-dominant, or all-quad). Surface meshes are used as input into the *Griddle* volume mesher **GVol**. The volume mesher uses the surface meshes to determine solid element size and type. Input surface mesh faces will appear as faces of solid elements in the volume mesh; thus, it is important to create good quality surface meshes before sending them to **GVol**. Input to **GSurf** must be one or more conformal (properly intersected) meshes or meshes that do not intersect at all.

The example in Figure 19 shows the result of application of **GSurf** to the meshes presented in Figure 18, right. The left image shows the conformal quad-dominant surface meshes remeshed to element size[5] 5-10m. The right image in Figure 19 shows same meshes but remeshed with local size control specified on the intersected (and extracted) patches of the input meshes (darker green and pink mesh patches in Figure 18, right). The local size for the patches was set to 0.5m while the global element size was set to 5-10m (see sections below on how specify local element size). **GSurf** creates a smooth transition between different element sizes. The gradation can be controlled.
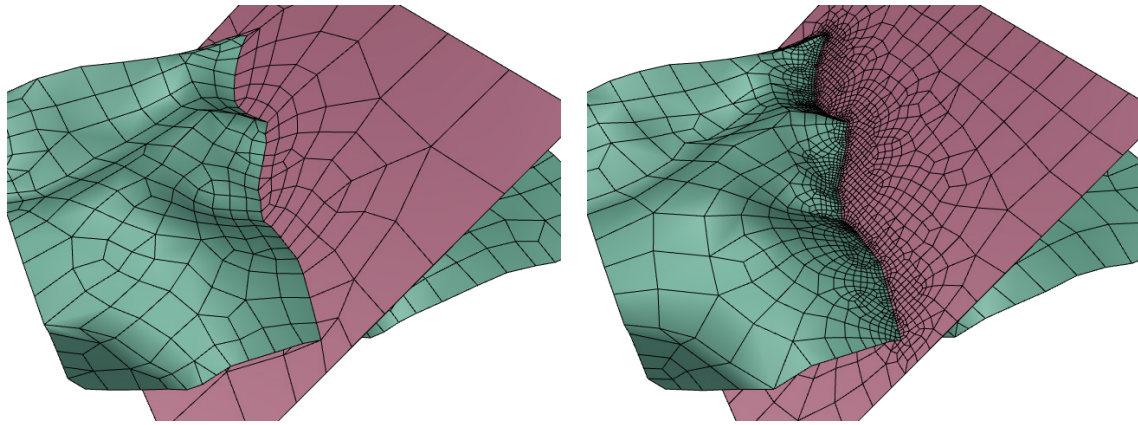
Figure 19: Examples of remeshing with *GSurf* without (left) and with local size control (right).

GSurf Options

Mode [*Tri (default), QuadDom, AllQuad*]
This option allows to choose the type of the output mesh:
- *Tri* produces an all-triangle surface mesh.
- *QuadDom* produces a quad-dominant surface mesh (contains a mix of triangles and quadrilaterals).
- *AllQuad* produces a pure quadrilateral surface mesh. In certain cases, it is impossible to create an all-quad mesh and **GSurf** may show an error. In this case use *QuadDom* mode instead.

---

[5] In this case, element size is determined by the average value between element edge sizes.

MinEdgeLength, MaxEdgeLength [*value > 0*]

These parameters control the output element size by setting minimum and maximum allowed edge sizes in the final surface mesh. To get uniform sizes, minimum and maximum edge size can be set to the same value. Edge size is specified in model units. It is important to set a non-zero minimum edge length to get a good quality output mesh.

Note that these values are not strictly enforced in the output mesh. However, **GSurf** attempts to achieve provided limits on edge size through multiple remeshing optimization passes (see *Optimization* below).

RidgeAngle [*value between* $0°$ *and* $90°$, *default* $= 20°$]

RidgeAngle, specified in degrees, controls the level of detail (the number of ridge lines) in the resulting mesh. The angle between mesh faces sharing an edge is termed a ridge angle (angle = 0° if faces are coplanar and 90° if faces are perpendicular). Ridge lines can be traced through the surface mesh by joining edges of faces that have ridge angles greater than the specified *RidgeAngle*. Using higher value for *RidgeAngle* results in less ridge lines (less detail) included in the final mesh. A lower *RidgeAngle* results in more detail included in the final mesh. Generally, *RidgeAngle* should be kept below 45°. The default value of 20° is a good compromise between mesh size and fidelity.

Advanced Parameters

MaxGradation [*value > 0, default = 0.1*]

This parameter controls the gradation of element sizes. A value close to 0 leads to a more gradual variation of mesh size (smoother), while higher values lead to more abrupt changes in element size.

Optimization [*value between* 0 *and* 10, *default* $= 5$]

This parameter controls the optimization of the mesh. A zero value makes **GSurf** skip the optimization step; the remeshing speed is the highest in this case, but the quality may be poor. From value 1 on, the optimizer algorithm uses several techniques to improve both the shape quality and the size quality of the elements, such as node smoothing, edge swapping, node insertion, and node removal. Level 5 is usually a good trade-off between quality and speed.

QuadWeight [*value between* 0 *and* 1, *default* $= 0.75$]

QuadWeight controls the preference of quadrilaterals vs. triangles in the output mesh. This parameter is used only in quad-dominant meshing mode. If *QuadWeight* = 0, quadrilaterals are never used. If *QuadWeight* = 0.5, quadrilaterals are used only when they improve the quality of the mesh. For values between 0.5 and 1, quadrilaterals are used more even if it leads to a lesser quality mesh. If *QuadWeight* = 1, the minimum number of triangles is used. This parameter is not the ratio between the number of quads and triangles. Furthermore, there is no linear relation between *QuadWeight* and the ratio between number of quads and triangles.

ShapeQuality [*value between* 0 *and* 1, *default* $= 0.7$]

ShapeQuality controls the trade-off between shape optimization and size optimization. The default value (0.7) gives a slightly stronger preference to the element shape quality over the size quality. For

example, an elongated surface mesh patch can be remeshed using few elongated triangles or quads (if *ShapeQuality* is close to 0) or it can be remeshed with many almost perfect triangles or quads (if *ShapeQuality* is close to 1).

OutputMesh [= *Separated (default), Merged*]
By specifying *OutputMesh = Separated*, **GSurf** will try to associate output mesh patches (sub-meshes) with the distinct input meshes and will place them into the layers they were originally obtained from. Patches that cannot be associated with a source mesh are placed into a layer "MISMATCHED_PATCHES". When *Merged* value is selected, a single output mesh will be generated and placed in the active layer.

DeleteInput [= *Yes (default), No*]
Specifying *Yes* in this parameter indicates that the original selected meshes will be deleted. If *No* is selected, the original meshes will remain intact.

Reset resets advanced parameters to the default values.

Additional Edge Size Control Options

In addition to the global values of min and max edge size specified in the parameters above (which are applied to all selected meshes), local overrides can be made to specify the desired element size for a surface mesh. Local values supersede the min and max edge sizes for the mesh.

Local element size for a mesh can be specified in one of two ways:

1. By specifying element size via a hyperlink available through object properties (Figure 20):
   - Select mesh
   - Open the *Properties* pane and click on the ellipsis (…) next to the **Hyperlink** property
   - In the **Hyperlink** dialog choose *Type: (other)* and type in the URL field "elemsize:*NumericValue*". Click **OK**. No spaces are allowed in the URL field.

2. By setting mesh name in the *Properties* pane to a numeric value representing the required element edge size (see image below).

Griddle first checks if element size is specified in the *Hyperlink* field and if nothing is found, it will check the Name field for a numeric value. Specifying local element size via hyperlink is preferred for a few reasons: clean syntax; the mesh name can contain digits that may be confused with element size; and, if the mesh name should not be modified.

Figure 20 shows two ways to specify local element size (*Name* field or *Hyperlink*). In this example only the yellow part of the mesh is assigned local element size, after which all meshes are remeshed with **GSurf**. The result of remeshing (bottom image) shows a much finer mesh at and around the yellow part, which corresponds to the provided element size.
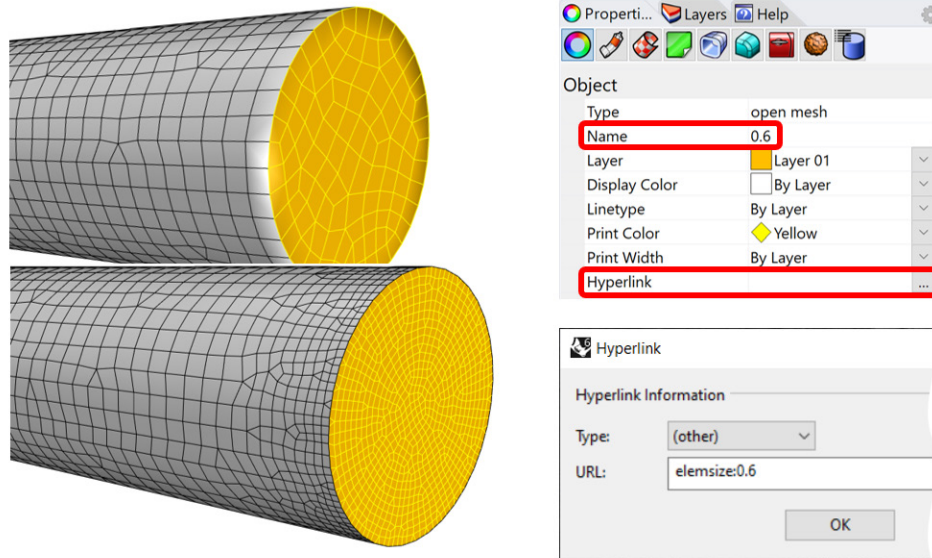
Figure 20: Two ways to specify local element size for surface meshes.

Similarly, local element size can be specified locally at a point, by creating a point (with *Rhino*'s **_Point** command) and assigning a numeric value to the point's hyperlink or name (as shown above). Such points must be coincident with the existing mesh vertices (enable vertex snapping in *Rhino* to snap the new point to a mesh vertex). If no element size is specified at a point, such a point will be treated as a <u>hard node</u> and will become a mesh vertex in the output mesh. Element size around this point will be determined automatically.

Figure 21: Specifying local element size at a point.

Hard Edges and Hard Nodes

Hard edges are edges that are preserved in a mesh during the remeshing process. Consider an example in Figure 22 (left), which shows two conformal surface meshes. If one of the meshes (for instance, the red mesh) is changed (for example, refined) while the other is unmodified, the conformity of the meshes will be broken. 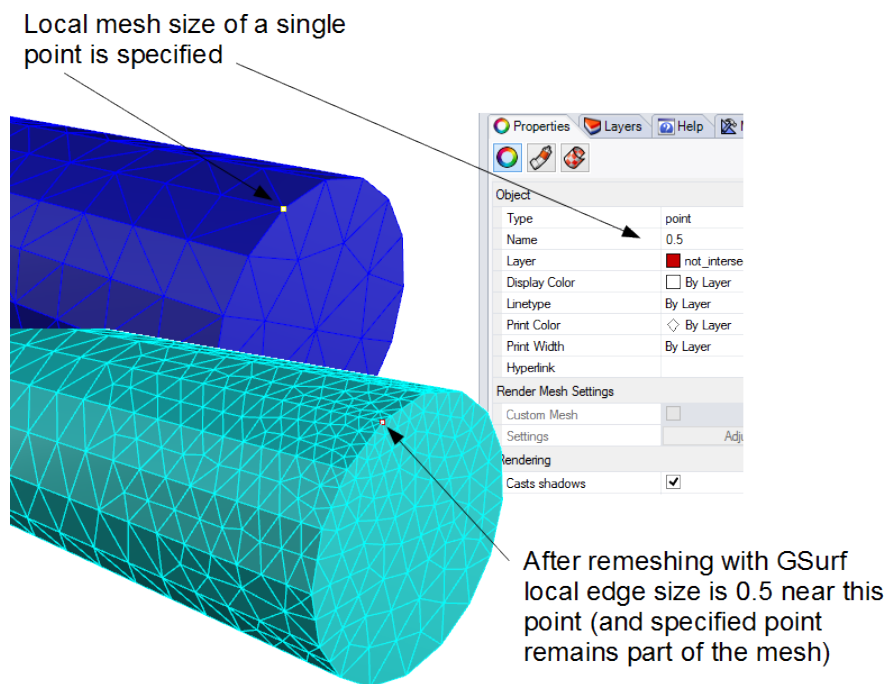**GSurf** offers a way to remesh a specific or all meshes while preserving conformity along mesh edges. This can be done by creating hard edges.

1. Use *Rhino*'s **_DupBorder** command on the red mesh, which will duplicate the mesh border (i.e., will create a curve connecting only nodes on the mesh boundary). Delete unnecessary segments of the curve (use the **_SubCrv** command) leaving only those segments that connect to the green mesh (as shown in Figure 22, left).
2. Select both the red mesh and the remaining part of the curve (polyline) and call **GSurf** with desired meshing parameters (in this case, smaller edge size). **GSurf** will recognize the polyline as a hard edge and will remesh the red mesh, taking the hard edge into account.

The remeshed red mesh will be finer, and both meshes will still be conformal.
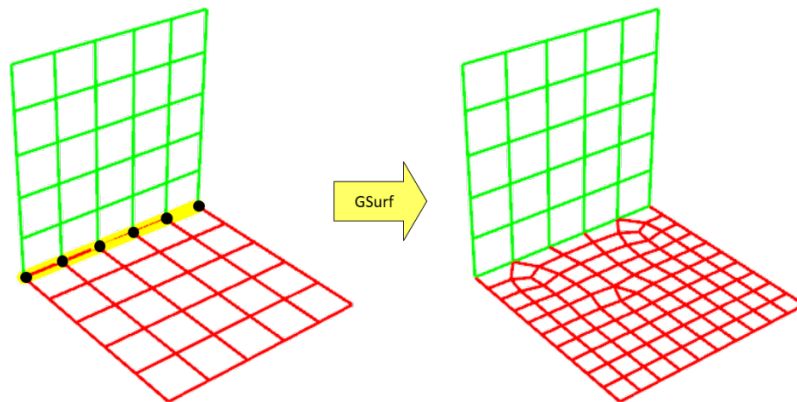
Figure 22: Specifying hard edges.

Similarly, a hard node can be defined to preserve a location of a vertex. Information about hard nodes is provided in the previous section.

GSurf outputs a text (ASCII) log file with information about input and output meshes. The log file uses the name of the *Rhino* project file (e.g. "RhinoProject.3dm") and adds "_GSurf.log" to it (e.g. "RhinoProject_GSurf.log"). It is output to the same directory where *Rhino* project is saved.

GVol – Unstructured Volume Mesher

GVol is *Griddle*'s unstructured volume mesher, capable of creating a tetrahedral or a hex-dominant mesh using selected surface meshes as boundaries (Figure 23). Surface meshes can be composed of triangles, a mix of quadrilaterals and triangles, or all quadrilaterals. The latter two must be used for generating hex-dominant grids. Tetrahedral grids can be generated from triangle, triangle-quad, or all-quad surface meshes (quadrilaterals are arbitrarily split along one of their diagonals into triangles).

There are two important requirements for *GVol* to successfully create a volume mesh:

- a combination of selected surface meshes must form a watertight boundary surrounding the entire volume of interest, and
- all intersecting or connecting surface meshes must be conformal.

Surface meshes can also separate discrete volumes within the larger volume. Surface meshes can also "float" inside the volume of interest or be partially connected to other surface meshes. All surface mesh faces, including "floating" surface meshes inside a volume, are included as "hard faces" in the final volume mesh. This means that all input surface faces will be present as faces of elements in the resulting volume grid.

Conformal surface mesh in Rhino   Volume element mesh (cutaway view)
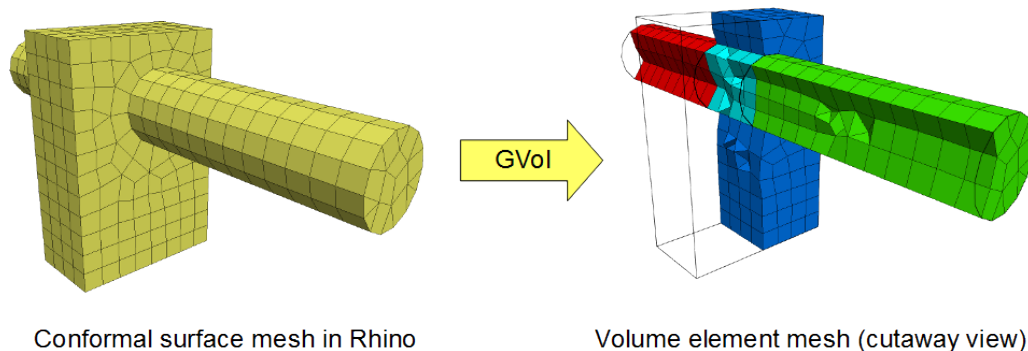
Figure 23: Volume mesh creation from surface meshes.

GVol requires input surface meshes to be properly connected. Duplicate, overlapping, or intersecting surface mesh faces are not allowed. Check the mesh for errors with *Griddle*'s *GHeal* tool and fix issues, if needed (note that remeshing may be required after fixing issues). If issues remain after using *GHeal*, consider intersecting surface meshes using *GInt* and specifying a higher tolerance and remeshing all meshes again.

If *GVol* encounters problems during meshing, traces of the problem areas will be placed into a *Rhino* layer "MESHING_ERRORS" (Figure 24). This layer will contain *Rhino* curves and points, outlining areas of surface meshes that caused problems. Typically, these problems are caused by the presence of naked edges (holes) or non-conformal (not properly intersected) mesh faces.

Figure 24: Traces of problem areas placed in "MESHING_ERRORS" layer.

GVol options

Mesh Settings
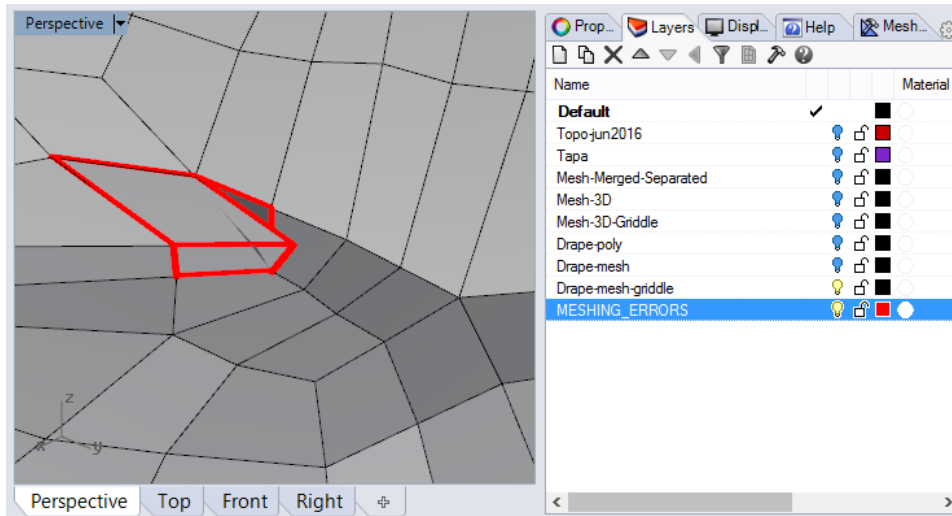
Mode [*Tet (default), HexDom*]
This option specifies the type of the output mesh:

- *Tet* (default) produces an all-tetrahedral volume mesh. Any quadrilaterals on surface meshes are converted to triangles in this case.
  HexDom produces a conformal hex-dominant volume mesh. Input surfaces must contain quadrilaterals for this option.

MaxGradation [*value > 0, default* = 0.5]
This parameter controls the gradation of element sizes from the size on the boundary to the preferred size defined by *TargetSize* inside the domain (far enough from the boundaries). A value close to 0 leads to a more gradual variation of size while higher values lead to more abrupt changes in element size.

TargetSize [*value > 0, default* = 0]
This parameter specifies the preferred element size inside the domain. The element size tends toward this value as elements get away from the boundaries. The default value of 0 indicates that the element size inside the domain is automatically determined based on the size of the elements on the boundary faces. Make sure that this parameter is meaningful and is not too small compared to the elements size on the boundary faces; otherwise, mesh generation may take a very long time as a large number of small elements will be generated inside the domain.

Optimization [*value between* 0 *and* 10*, default* = 5]
This parameter controls the optimization of the mesh. A zero value makes the mesher skip the optimization step. The speed is the highest, but the quality may be poor. From value 1 on, the

optimizer algorithm uses several techniques to improve both the shape quality and the size quality of the elements, such as node smoothing, edge swapping, node insertion and node removal. Level 5 is usually a good trade-off between quality and speed.

ShapeQuality [*value between* 0 *and* 1, *default =* 0.7]
This parameter controls the trade-off between shape optimization and size optimization. The default value (0.7) gives a slightly stronger preference to the element shape quality over the size quality. For example, an elongated volume region can be filled with few elongated elements (if *ShapeQuality* is close to 0) or it can be filled with many almost perfect elements (if *ShapeQuality* is close to 1).

IniErrorCheck [= *Yes, No*]
IniErrorCheck specifies if an error check for input surface meshes should be executed to detect potential problems with naked edges and/or clashing faces. The error check is executed before volume meshing begins, and if problems are found, a warning message will be shown. The initial error check can be turned off by setting this option to *No*. *Griddle* remembers this option even after *Rhino* is closed, therefore if *IniErrorCheck* is set to *No*, next time *Rhino* is launched, the initial error check will not run. To re-enable the check, set it to *Yes*. It is recommended to keep *IniErrorCheck = Yes* to be able to see warnings about input mesh problems. Changing this option does not affect other error checks during and after volume mesh generation.

Output Format Parameters

OutputFormat [= *FLAC3D, 3DEC_5x, 3DEC_7x, ABAQUS, ANSYS, NASTRAN, LS-DYNA, VRML, CSV*]
OutputFormat specifies format in which the resulting volume grid file should be saved.

FormatType [= *Binary, Text*]
This option appears only if the user selects *OutputFormat = FLAC3D* or *3DEC_7x* (i.e., *3DEC* v7.0 or higher). The option sets how to save the output file: in binary or text (ASCII) format. Saving and loading (reading) files in binary format is much faster compared to the text format, but binary files are not human-readable. All other *OutputFormat* choices allow saving in text format only.

BlockType [= *Rigid, Deformable*]
This option appears only if user selects *OutputFormat = 3DEC_7x* (i.e., *3DEC* v7.0 or higher). It specifies if each element of the output volume mesh should be represented as a rigid block (*BlockType = Rigid*) or as a deformable zone (*BlockType = Deformable*) in *3DEC*. In the latter case, watertight volumes in the model (closed volumes separated by surface meshes) will be represented as rigid blocks filled with deformable zones.

AutoOutputName [= *N/A (default), UserDefinedName*]
This option sets a string that will appear in the output filename along with other identifiers. For example, for a *Rhino* project named "RhinoProject.3dm", a *3DEC* rigid blocks output in binary format will be named: "RhinoProject_*UserDefinedName*_Rigid_Binary.3dgrid"). If the user specifies a string in this option, the **Save As** dialog will not appear, and the output file will be named automatically as

described above. If *AutoOutputName* is not specified (N/A), the **Save As** dialog will appear in order to obtain the output file name and location.

- Note that the value for this option is not saved: each time user clicks on this option, a new string must be typed. To erase the existing string in *AutoOutputName*, click on the option and press **Enter** without typing anything; this will cause the **Save As** dialog appear when executing *GVol*.
- The output file will be saved in the same location as the *Rhino* project file. If a file with same name already exists in that location, it will be overwritten.
- This option allows running *GVol* in automatic mode without user interaction (i.e., the need to type file name and press **OK** / **Cancel** in the **Save As** dialog). This is particularly useful when invoking the *GVol* command from *RhinoScript*, *PythonScript*, or other *Rhino* tools and plugins.

Important Notes

If *OutputFormat* = *3DEC_7x* and *BlockType* = *Deformable*, block joints will not be created along floating surface meshes located (fully or partially) within watertight volumes that will become rigid blocks (Figure 25). Such floating surface meshes (meshes with naked edges) do not split or define separate rigid blocks and currently cannot be represented as joints in *3DEC* (though, they can be represented as interfaces in *FLAC3D*). To check for the presence of surface meshes with naked edges, set *IniErrorCheck* = *Yes* and/or use the **_GHeal** command.
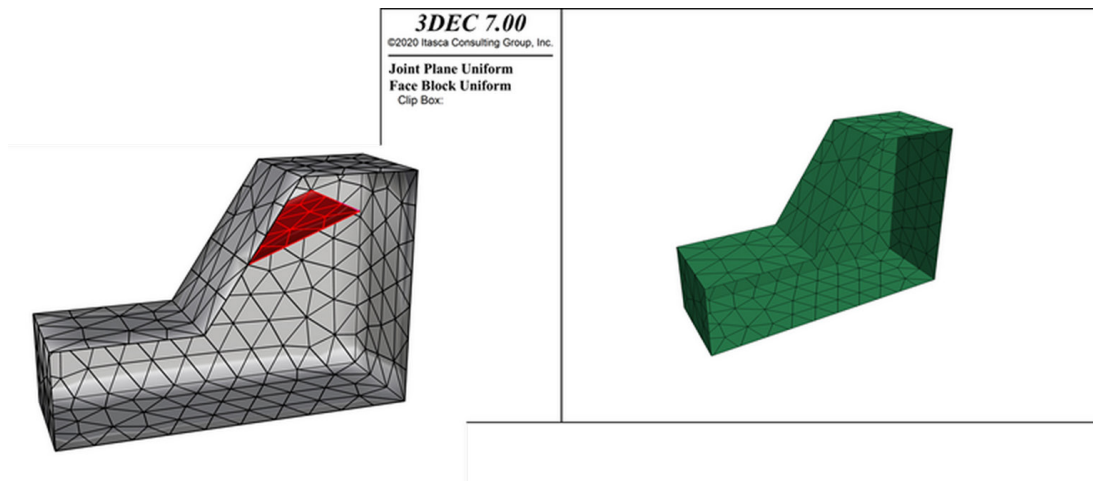

Figure 25: *Rhino* **model (left) with floating surface (red) and** *Joint* **plot in** *3DEC* **representing the rigid block (right). The red surface does not become part of the rigid block (or a joint) in** *3DEC*.

GVol outputs a text (ASCII) log file. The log file uses the name of the *Rhino* project file (e.g., "RhinoProject.3dm") and adds the following to the filename: "_GVol" + format type (Text/Binary, if applicable) + Output Format (e.g. ".f3grid", ".inp") + ".log" (e.g. "RhinoProject_GVol_Binary.f3grid.log"). The log file is output to the same directory where the *Rhino* project is saved.

Additional *Griddle* Utilities for Working with Surface Meshes

GHeal – Tools for Surface Mesh Repair

GHeal is a *Griddle* utility that can be used to identify certain surface mesh problems and fix them. In automatic mode, **GHeal** attempts to fix such issues as: patch mesh holes, patch (mend) cracks (elongated fissures on the external boundary of a surface mesh), clean clashing faces, remove small protrusions (a small number of mesh faces attached to the main mesh in non-conformal way), split non-planar quadrilateral faces, align mesh normals, and clean (cull) redundant nodes and detached faces. Each of these operations are referred to as **GHeal** functions. In certain cases, running all these functions at once helps cleaning up and fixing a mesh. In other cases, only specific functions should be executed to clean up a mesh, as some other functions may distort the mesh or provide undesirable results. In general, **GHeal** functions (executed collectively or individually) often identify and clean mesh problems more efficiently than user efforts to fix a mesh manually or by using available *Rhino* tools.

GHeal also has a *Custom* operation mode which calls *Rhino*'s command **_MeshRepair**. This command allows for more precise repairs of a mesh, e.g., patching a single hole where the user must select the boundary of the hole; **GHeal**'s *AutomaticHeal* operates on the whole mesh. It attempts to find all issues and repair parameters automatically. Refer to *Rhino* help documentation for more information about the **_MeshRepair** command.

GHeal Options

ShowErrors
This option searches selected surface meshes for naked edges and clashing faces and plots traces of such edges and faces.

Naked edges are face (or surface) edges that are not joined to other faces (or surfaces). In other words, only a single face (surface) shares/contains such edge. Mesh boundaries and discontinuities contain naked edges. They can also be found along disjointed faces and may not be visible (faces may appear to be connected that are not).

Clashing faces are pairs of faces which are partially overlapping or intersecting in a non-conformal way.

When clicking on this option, traces of naked edges are placed into the *Rhino* layer "NAKED_EDGES" and traces of the clashing faces are placed into the layer "CLASHING_FACES". Examining polylines placed in these layers reveals potential mesh issues.

Note that surface meshes within a watertight domain can have naked edges, for example, be partially connected to domain boundaries or "float" within the volume (also see information on **GVol**). However, the presence of naked edges on domain boundary meshes due to holes, disjointed faces, or

other discontinuities, as well as presence of clashing faces, usually indicates problems with surface meshes. Such problems must be addressed before volume meshing.

CustomHeal

This option calls *Rhino*'s command **_MeshRepair**, which checks meshes and fixes problems manually (one by one). Refer to *Rhino* help for the description of the **_MeshRepair** command.

AutomaticHeal

AutomaticHeal mode contains two options: *IssuesToFix* and *Parameters*. *IssuesToFix* specifies types of issues to work on and *Parameters* specify corresponding function parameters. They are described below in detail.

IssuesToFix

This option specifies types of issues (functions) that **GHeal** can operate on in *AutomaticHeal* mode. Any combination of the *issues* can be selected for **GHeal** to work on (by default all are selected). Every selected function is executed in the sequence it appears in the menu.

All and *None* options allow to select all or deselect all issues in the list.

FillHoles

This function fills all holes within the selected meshes (Figure 26). The function uses the parameter *MaxHoleArea*, which is the maximum area of a hole to be filled. A hole will be filled if (i) the area of the hole is smaller than *MaxHoleArea* and (ii) the area of the hole is smaller than half of the total area of the entire mesh. Remeshing may be needed after filling large number of holes.
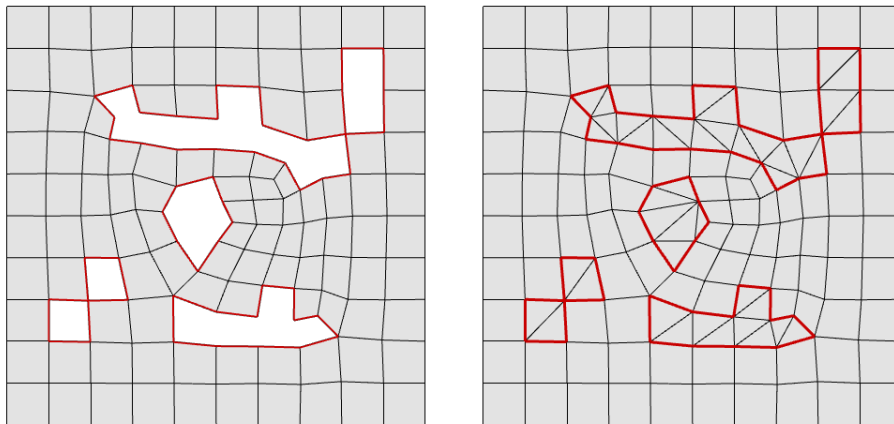

Figure 26: *GHeal*'s function *FillHoles* fixes various holes in the surface mesh. Red lines show naked edges (left) and are provided for reference only in the right image.

MendCracks

This function mends ("stitches") all cracks on the edges of the selected meshes (Figure 27). A crack differs from a hole by that it is connected to the external boundary of the mesh. The function uses the parameter *MaxCrackOpening*, which is the maximum distance between the edges of a crack.

Crack "stitching" starts at the tip of the crack and proceeds as long as (i) the opening angle at the crack tip is <= 90° and (ii) the maximum distance between the edges of the crack is smaller than *MaxCrackOpening*. Remeshing may be needed after fixing large cracks.
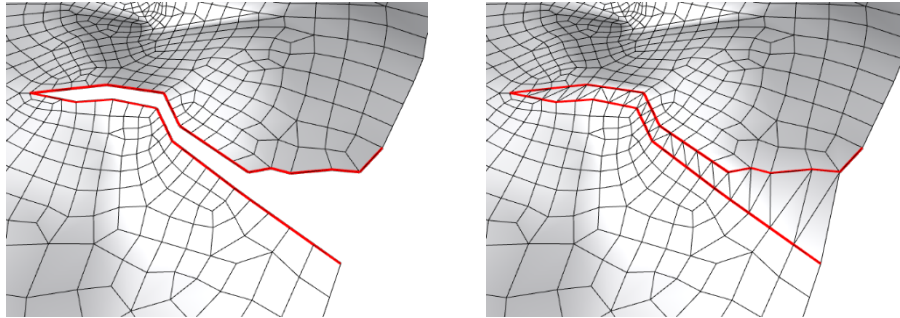

Figure 27: *GHeal*'s function *MendCracks* "stitches" a crack on the boundary of a surface mesh. Red lines indicate naked edges (left) forming the crack and are provided for reference only on the right image.

FixClashingFaces

This function fixes clashing mesh faces (faces which intersect or overlap) within each selected mesh (Figure 28). *FixClashingFaces* does not use **GInt** and attempts to intersect/split faces locally. *FixClashingFaces* will not fix non-conformal and overlapping faces from different intersecting meshes. To properly intersect distinct meshes, use the mesh intersector **GInt**.
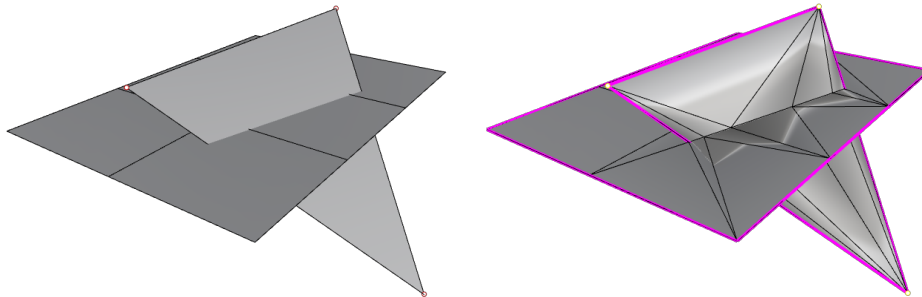

Figure 28: *GHeal*'s function *FixClashingFaces* locally intersects/splits clashing faces.

Note that in certain cases functions *FillHoles* and *MendCracks* may introduce clashing faces if newly built faces intersect with existing faces. Using *FixClashingFaces* together with these mentioned functions helps resolve this problem. Remeshing may be needed after fixing clashing faces.

RemoveProtrusions

This function removes mesh faces (patches) protruding from the main surface of the mesh. An example of such protrusion is shown in Figure 29. The function uses the parameter *MaxProtrusionPerimeter*, which is the maximum perimeter of the protruded mesh patch. The protruded mesh patch will be removed if it is joined to the rest of the mesh in non-manifold manner and if (i) the total perimeter of the protruded patch is smaller than *MaxProtrusionPerimeter* or (ii) there are two or less faces in the protruded mesh patch.
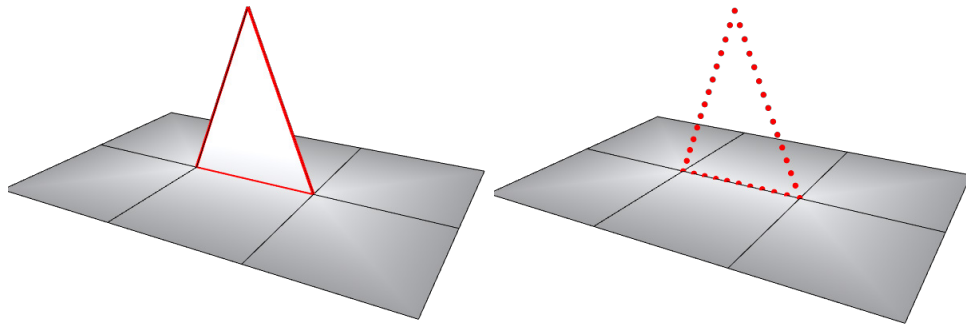
Figure 29: *GHeal*'s function *RemoveProtrusions* removes a single face which "sticks out" from the main mesh.

SplitNonPlanarFaces

This function splits non-planar quadrilateral faces into pairs of triangular faces. The non-planar quadrilaterals are divided along the short diagonal.

CullRedundancies

This function culls unused mesh points and degenerated faces and combines points that are too close to each other. The function uses the parameter *MaxNodeDistance*, which is the maximum distance between two mesh nodes below which the nodes will be considered coincident.

AlignNormals

This function aligns normals in each component of the mesh and it works on compound meshes consisting of joined/welded sub-meshes. This function is more robust than *Rhino*'s **_UnifyMeshNormals** command when working with compound meshes.

Parameters [= Automatic (default), Custom]

This option allows specifying parameters for the functions <u>selected</u> in the *IssuesToFix* menu. *Parameters = Automatic* automatically calculates and uses optimal parameters for the functions listed above. *Parameters = Custom* allows user to manually specify parameters for **GHeal**'s functions. Possible options are:

MaxHoleArea [value > 0, default = 1]

This parameter defines, in model units, the maximum area of a hole to be filled.

MaxCrackOpenning [value > 0, default = 0.01]

This parameter defines, in model units, the maximum distance between the edges of a crack in model units.

MaxProtrusionPerimeter [value > 0, default = 0.1]

This parameter defines, in model units, the maximum perimeter of the protruded mesh patch in model units.

MaxNodeDistance [value > 0, default = 0.001]

This parameter defines the tolerance or the maximum distance between two mesh nodes below which the nodes will be considered coincident. Model units are used.

GExtract – Tools for Extraction of Surface Meshes

GExtract is a *Griddle* utility that allows extracting parts of surface meshes based on user-specified criteria. As surface meshes approximating real engineering and geological structures may be rather complex, there is often a need to extract (or separate) parts of the meshes either to remove them or to assign specific properties in preparation for volume meshing. *GExtract* has five different modes of operation to assist with that.

For the first two modes described below, *GExtract* extracts faces based on a break angle and whether to stop the extraction at non-manifold edges (a common edge that is shared by more than 2 faces). In these modes, the command is similar to *Rhino*'s command **_ExtractConnectedMeshFaces,** except the *GExtract* utility allows for a consistent manner of selecting faces of non-manifold meshes and stopping only if the angle exceeds the break angle and/or at non-manifold edge.

GExtract Options

SingleSurface

In this mode *GExtract* extracts single manifold sub-mesh containing a selected face. When using this mode, user first must select a mesh face from which the rest of the selection will propagate (Figure 30). After that a dialog will be shown (Figure 31, Figure 32) that requests criteria for stopping selection. When the user clicks **OK**, all the selected faces will be separated as a new mesh (mesh properties will match the original properties).
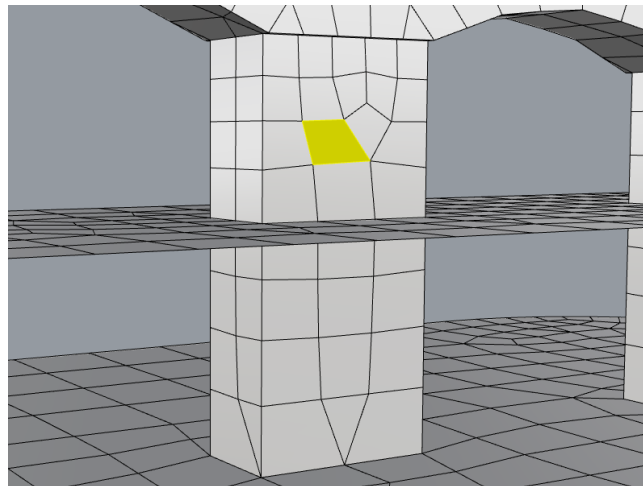

Figure 30: Selecting an initial face for *GExtract*'s *SingleSurface* extraction mode.

The **Non-Manifold Extract Faces** dialog sets two criteria for stopping selection:
- By a break angle (*Max break angle*: 0-180°) or
- when the selection hits a non-manifold edge (*Break at non-manifold edges*).

The first option specifies the angle between joined faces (in degrees). Specifying 0° will only select joined faces that are coplanar with the face picked. If the second option is disabled, *GExtract* will

propagate its selection of faces at non-manifold edges to those faces having the smallest angle with the currently processed faces (i.e., **GExtract** will follow the smoothest path at non-manifold edge).

The second option allows stopping the selection at non-manifold edges. Figure 31 and Figure 32 show the results when breaking at non-manifold edges is enabled or disabled.
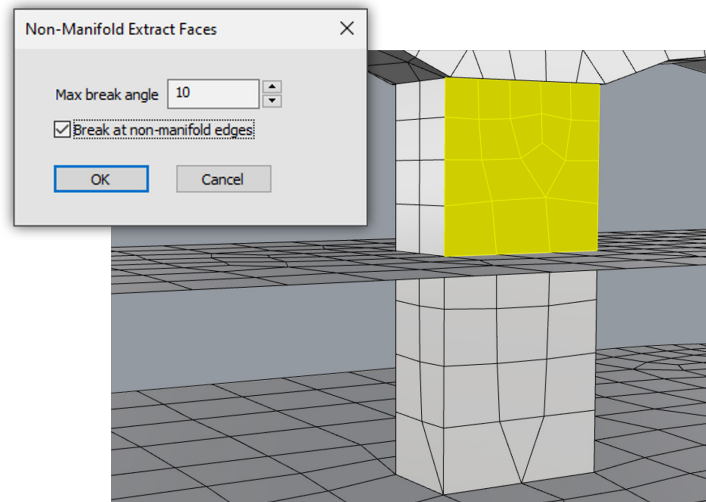

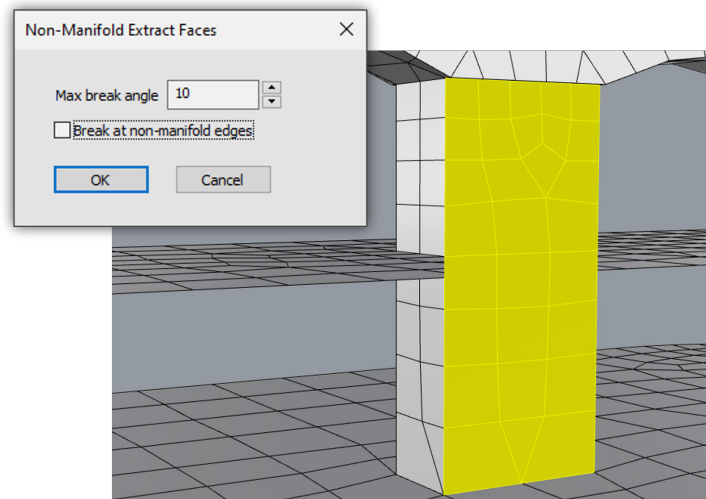**Figure 31: Face selection in the case where *Break at non-manifold edges* is enabled.**

Figure 32: Face selection in the case where *Break at non-manifold edges* is disabled.

AllSurfaces

This mode extracts all possible manifold sub-meshes from selected surface meshes. This option specifies break angle only (0-180°). The extraction of sub-meshes occurs when either angle between neighboring faces exceeds the specified break angle or if a non-manifold edge is encountered. Note that initial faces are picked automatically; the user does not need to pick any faces.

The example in Figure 33 and Figure 34 shows the use of *AllSurfaces* mode with break angle 60°. A single non-manifold conformal mesh (Figure 33) is split into multiple manifold conformal meshes (Figure 34). Afterwards, the extracted meshes are assigned different colors using the **ColorizeObjects** tool.
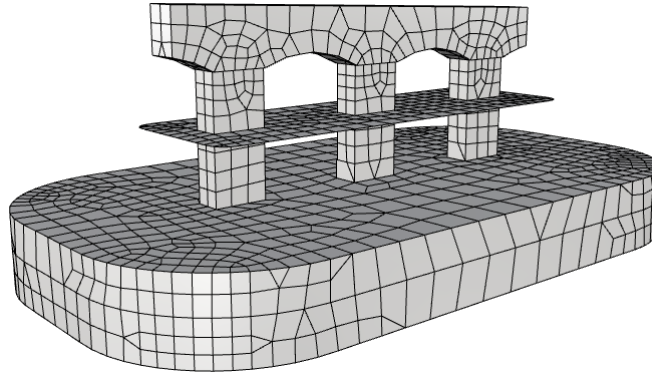
Figure 33: Initial single non-manifold conformal mesh (some internal meshes are not visible).
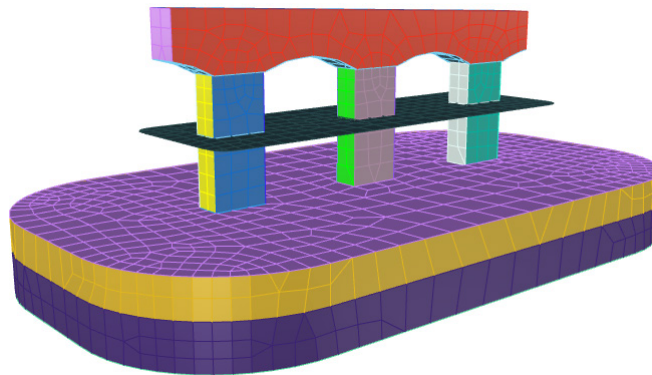

Figure 34: The result of application of *GExtract* with *AllSurfaces* mode to the initial mesh.

When using *AllSurfaces* mode, the angle between faces is calculated using the angle between face normals. Therefore, it is important that all mesh normals are aligned to have proper mesh extractions. If mesh extraction leads to unexpected results, align mesh normals using **GHeal**'s *AlignNormals* function or *Rhino*'s **_UnifyMeshNormals** command (it may not work properly for compound/joined meshes).

BoundaryFaces

This mode extracts only those faces that are attached to naked edges (i.e., mesh boundaries). Extracted faces are combined into a separate mesh that is assigned the same layer and name as the original mesh but a different color. This option can operate on multiple meshes simultaneously. An example of extraction of boundary faces from two meshes is shown in Figure 35.
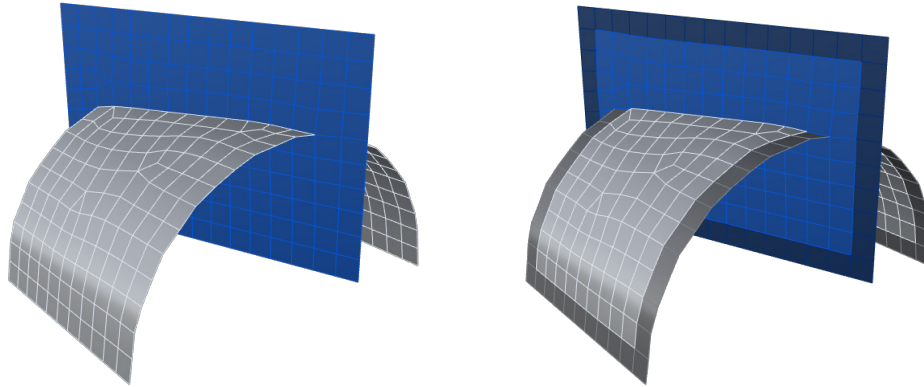
Figure 35: Extraction of boundary faces using *GExtract's* *BoundaryFaces* mode.

NonManifoldFaces

This mode extracts only those faces that are attached to non-manifold edges if any are present in the mesh. Extracted faces are combined into a separate mesh that is assigned the same layer and name as the original mesh but a different color (Figure 36). The option can operate on multiple meshes simultaneously. This option may be used to refine a mesh around non-manifold edges by using the surface remesher **GSurf** (for example, (i) assign a custom element size to the extracted mesh and remesh everything or (ii) specify hard edges along extracted mesh boundaries using the **_DupBorder** command and then remesh only the extracted mesh).
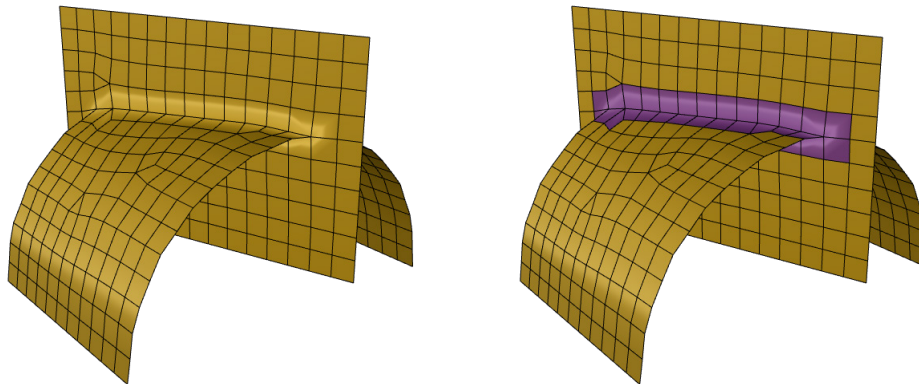

Figure 36: Extraction of faces attached to non-manifold edges using *GExtract's* *NonManifoldFaces* mode.

SurfacesWithinSolid

This mode extracts pieces of meshes that are located within a specified close volume solid (e.g., a box, cylinder, etc.) Extracted mesh pieces have the same properties as the original meshes but are assigned a slightly different color. After the operation completes, extracted mesh pieces are selected so it is easier for the user to do other operations with them (e.g., hide, delete, move, remesh, etc.) It is advisable to keep the solid in a separate layer, so it can be easily hidden while keeping the extracted meshes selected (turn off the whole layer). The option can operate on multiple meshes simultaneously. This option may be used to change element size in extracted meshes by using the surface remesher **GSurf** (for example, (i) assign custom element size to the extracted mesh and remesh everything or (ii) specify hard edges along extracted mesh boundaries using the **_DupBorder** command and then remesh only the extracted mesh).

Figure 37 shows an example of extraction of sub-meshes located within a cylinder (the solid). The cylinder is represented as BRep object, and it is not a mesh. First, user must select all meshes to process and then select the solid. After **GExtract** finishes operation, the extracted meshes are highlighted (top-right image). If the cylinder was placed into a separate *Rhino* layer, it can be easily hidden from the view by turning off the corresponding layer. The extracted meshes stay selected (bottom-left image). At this point user may change properties of the selected meshes or do other operations with them.
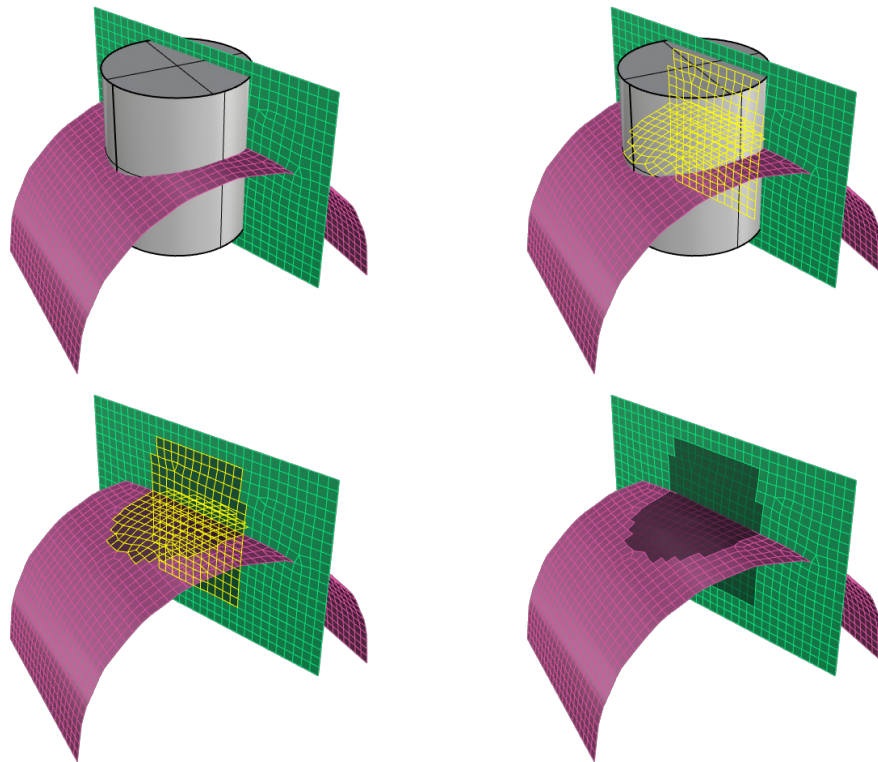

Figure 37: Process of extraction of mesh faces located within a solid by using *GExtract*'s *SurfacesWithinSolid* mode.

GExtend – Tools for Surface Mesh Extension

GExtend is a *Griddle* utility that allows extending a surface mesh by adding new faces to it. New mesh faces are created along a selected part of the mesh boundary or along the whole boundary. After that they can be joined to the original mesh or kept as a separate mesh (the original mesh is not changed). **GExtend** attempts to preserve average tangent (or average local mesh curvature) when building new faces. The average tangent is calculated based on the initial mesh faces attached to the selected part of the boundary.

The most important parameter of **GExtend** is the extension length, which specifies edge size for newly built faces (along the extension direction). The extension length should be comparable to the size of faces in the initial mesh at the selected part of the boundary. If the extension length is significantly larger than the typical initial face/edge size, the extended mesh may be of poor quality or even be invalid due to the requirement of preserving average tangent.

After using the **_GExtend** command, remeshing of the extended surface mesh may be necessary to improve mesh quality and to achieve desired element size.

GExtend has three modes of operation: *ExtendSelectedBoundary*, *ExtendAllBoundaries*, and *FreeExtend*. For the first two modes, the command extends the boundary by a given distance while the third mode allows the user to do "free" extension of the selected boundary by moving it to any location.

When extending a mesh along a part of the mesh boundary, all mesh boundaries are highlighted in purple for easier selection (Figure 38). With boundaries highlighted, the user selects a desired region as shown in Figure 38. Additional pieces of the boundary can be added (or removed) by pressing and holding **Shift** (**Ctrl**) and the left mouse button. Make sure to select a continuous piece of the boundary as **GExtend** operates on single continuous segment of the boundary.
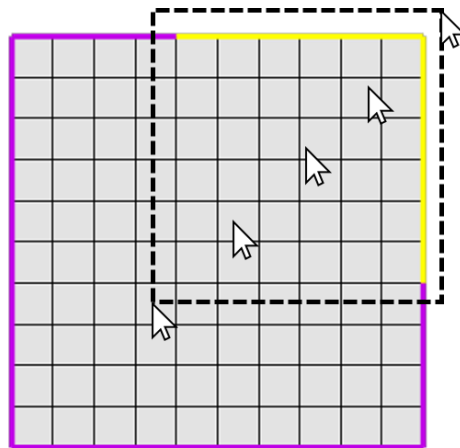

Figure 38: *GExtend* highlights mesh boundary in purple and allows selecting a piece of the boundary (in yellow).

GExtend Options

ExtendSelectedBoundary

In this mode, **GExtend** extends the mesh along the selected part of mesh boundary by a given distance (Figure 39). Two parameters should be specified for the operation to begin:

ExtendLength

This parameter specifies, in model units, the extension length. It is the same as the edge length of newly built faces in the extension direction.

MeshType [= *Merged* (*default*), *Separated*]

This parameter specifies if the newly built (extended) mesh patch should be merged with the initial mesh or kept separate.
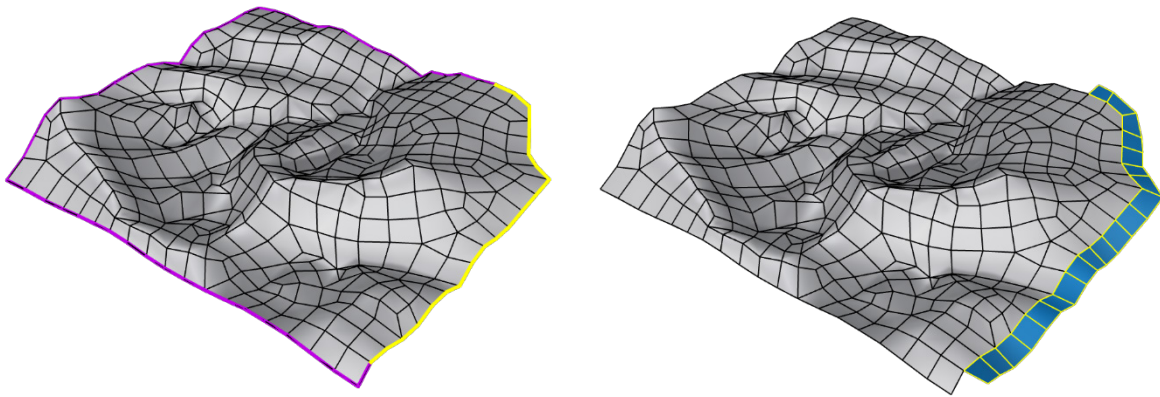

Figure 39: Extension of a mesh along part of its boundary using *ExtendSelectedBoundary* mode.

ExtendAllBoundaries

This mode expands the initial mesh(es) along all its boundaries by a given distance (Figure 40). Multiple meshes can be selected and boundaries of each mesh will be extended. The options in this mode are the same as in *ExtendSelectedBoundary* mode.
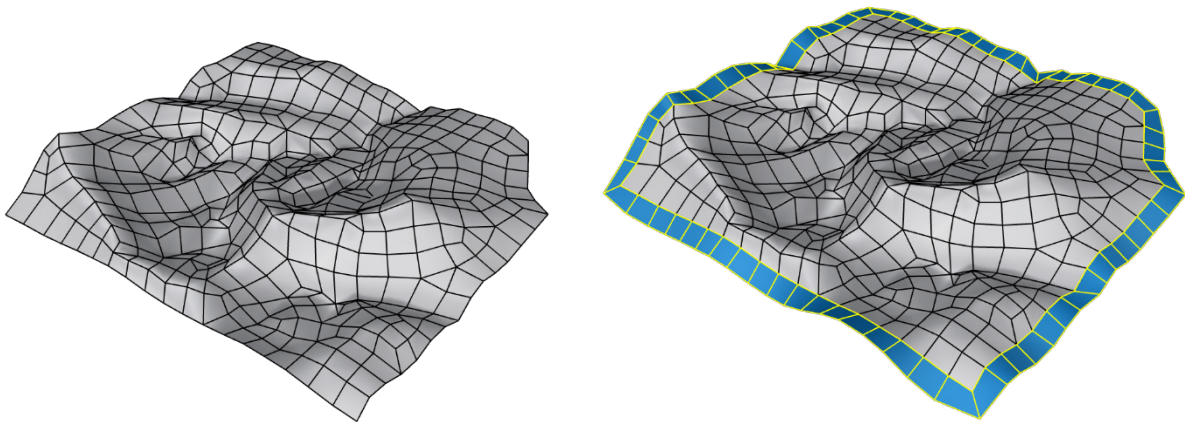

Figure 40: Expansion of a mesh along all its boundary using *ExtendAllBoundary* mode.

FreeExtend

This mode extends a mesh at a selected part of the boundary to a custom position by moving (dragging and dropping) the selected part (Figure 41). The movement starts at a node in the middle of the selected part and stops at a point where user "drops" it. The extended part of the mesh is merged with the initial mesh.
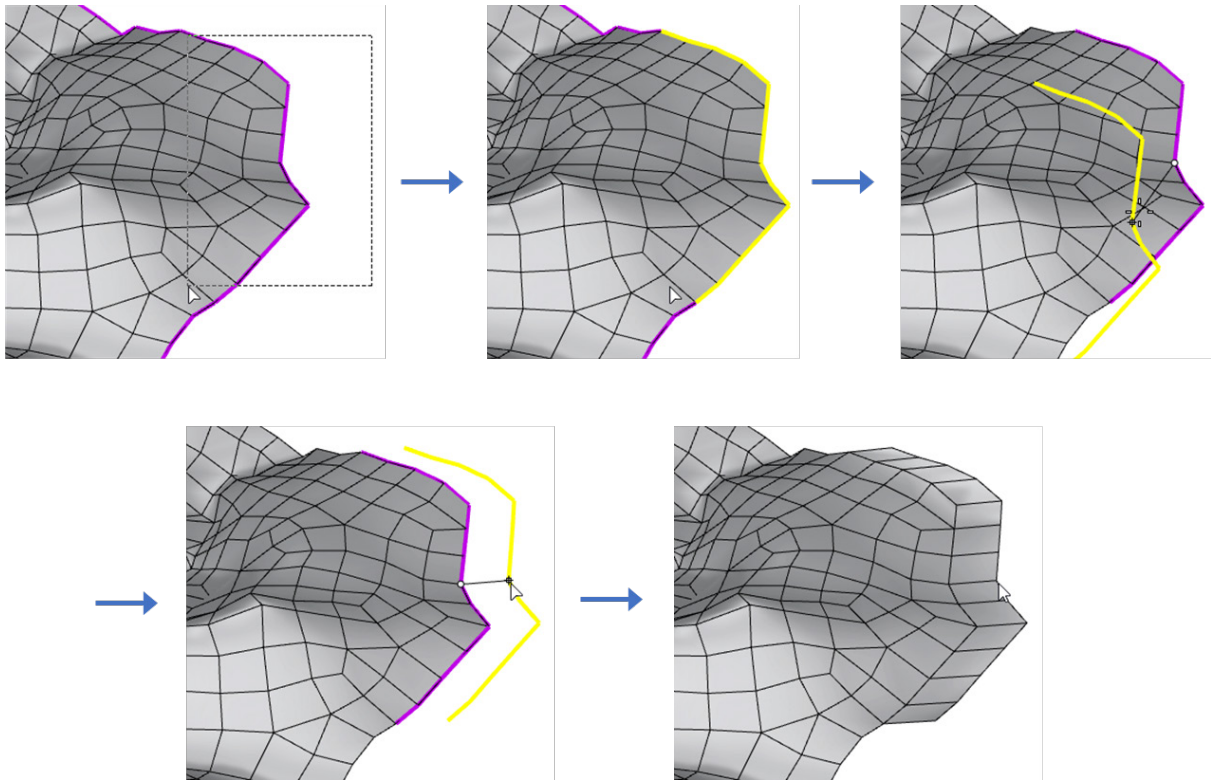

Figure 41: Process of selecting and dragging a part of a mesh boundary using *FreeExtend* mode.

GExtrude – Tools for Surface Mesh Extrusion

GExtrude is a *Griddle* utility that allows extruding a surface mesh to a bounding surface to create a watertight mesh domain. The extrusion is done along the mesh external boundary and follows these steps.

1. Project each node on the boundary of the initial surface mesh onto the bounding surface (using shortest distance).
2. Connect the projected nodes to each other and to the corresponding original nodes to form side surfaces.
3. Mesh the side surfaces.
4. Fill the area within the projected boundary on the bounding surface with a mesh.

This approach of mesh extrusion onto a bounding surface is designed to quickly create a watertight modeling domain from a given surface mesh (e.g., a topographic mesh). It works well when the bounding surface is planar or nearly planar. The approach is not designed for exact mapping of the initial surface mesh onto a bounding surface and it does not guarantee that mesh projection will exactly conform to the bounding surface if it is non-planar. The approach requires that the initial surface mesh is simply connected— meaning it cannot contain holes, naked edges, or internal boundaries (check mesh quality and fix issues with **GHeal**, if needed).

GExtrude has three basic options: *ExtrdMeshType*, *MeshOutput* and *MeshMode*. Several additional options vary depending on the *MeshMode*.

GExtrude options

ExtrdMeshType [= *Tri* (*default*), *QuadDom*]
This parameter specifies the type of surface mesh used in the extruded mesh: *Tri* produces an all-triangle surface mesh, *QuadDom* produces a quad-dominant surface mesh (contains a mix of triangles and quadrilaterals).

MeshOutput [= *Merged* (*default*), *Separated*]
This parameter specifies if the extruded mesh should be merged with the initial mesh into a single mesh object or kept separate.

MeshMode [= *Unstructured* (*default*), *Structured*]
This parameter specifies the mode of the output mesh. When it is set to *Unstructured* mode, two additional parameters will display:

MinEdgeLength, MaxEdgeLength.
These parameters control the resulting minimum and maximum edge size in the final surface mesh. To get uniform sizes, minimum and maximum edge size can be set to the same value. Edge size is specified in model units. It is important to set non-zero minimum edge length to get a good quality output mesh.

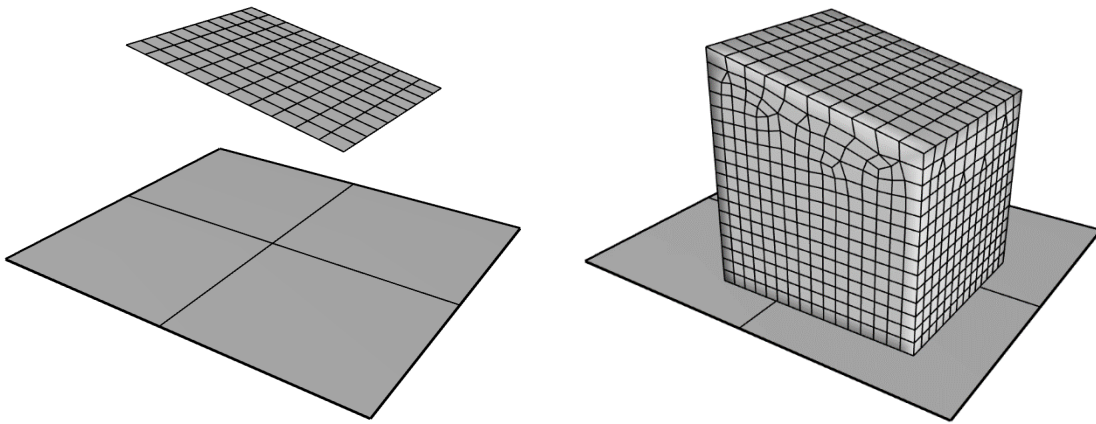Figure 42 demonstrates how the command extrudes a surface mesh onto a bounding surface when *MeshMode = Unstructured*.

Figure 42: *The initial mesh and bounding surface (left) and the results of mesh extrusion to form a watertight domain.*

When *MeshMode* is set to *Structured*, an additional option *StructuredParams* appears:

StructuredParams [=*FixNumberElements* (*default*), *FixEdgeLength*]
If *FixNumberElements* is chosen, the number of elements along the extrusion direction is fixed and is defined by the option *NumElements*, which the user must specify.
If *FixEdgeLength* is chosen, the element length along the extrusion direction is fixed and is defined by the option *EdgeLength*, which the user must specify.

Figure 43 demonstrates the results of the extrusion when *MeshMode = Structured* and *StructuredParams = FixNumberElements* (left image) and when *StructuredParams = FixEdgeLength* (right image).
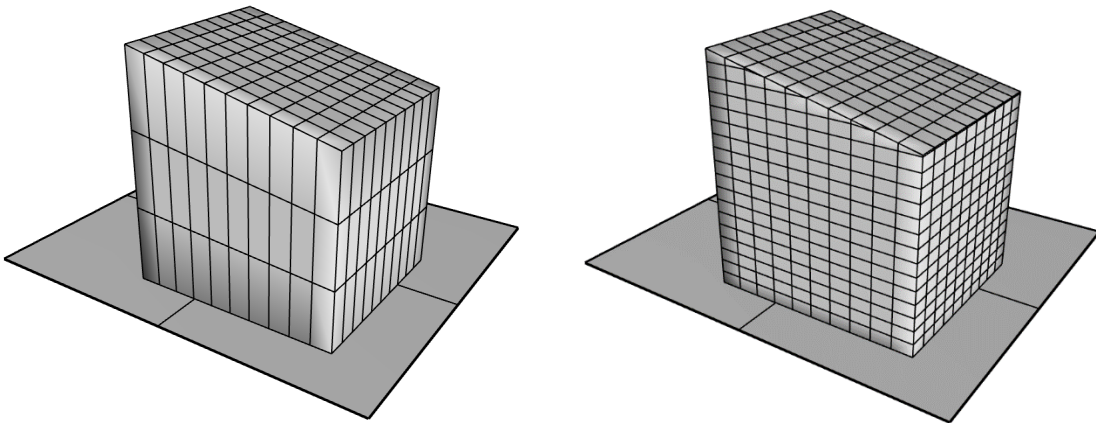

Figure 43: Results of mesh extrusion when using *Structured* mode and fixing the number of elements (left) or the edge length (right) along the extrusion.

Joining Non-Manifold Surfaces

_NonManifoldMerge is a *Rhino* command that joins several manifold or non-manifold surfaces and polysurfaces into a single non-manifold polysurface[6]. It is provided in *Griddle* for convenience as it can be used in the process of preparing surface meshes for volume meshing.

If NURBS, SubD, or BRep analytical surfaces are available, the user may create an initial conformal mesh of all objects by first joining them with **_NonManifoldMerge** and calling the **_Mesh** command. When creating an initial mesh for a single (even complex) object, *Rhino* builds a fully conformal mesh (as opposed to the case of meshing multiple objects separately), so the mesh intersection step (as in Figure 16) may be skipped. The next step is to remesh the initial mesh with **GSurf** and then create a volume mesh.

This procedure avoids problems related to using improper tolerance during the mesh intersection step (**GInt**) and thus obtains accurate results faster. However, the drawback of this approach is that the output surface mesh is now a single object (it can be split using **GExtract**). A simple example of such procedure is shown in Figure 44 and a detailed example is provided in *Griddle 2.0 Tutorial Examples*.
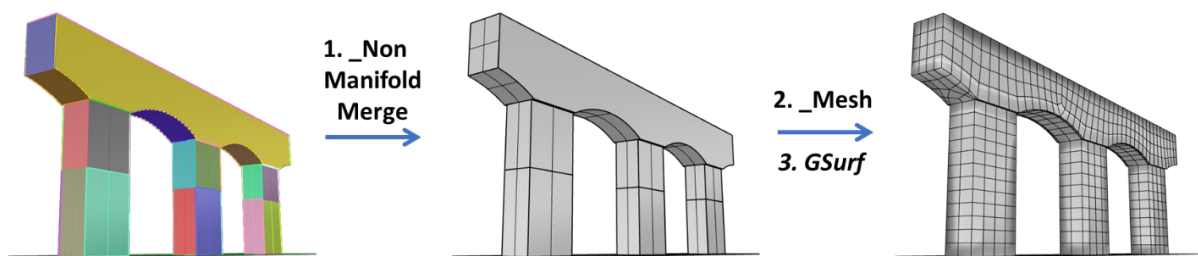

Figure 44: Three step process to create a single mesh out of multiple surfaces/polysurfaces.

---

[6] In this context, manifold means that a surface edge is shared by at most two surfaces. In contrast, consider two intersecting planar surfaces and represented as a single polysurface object. Now the edge along the intersection line is connected to 4 sub-surfaces. Thus, the whole polysurface object will be non-manifold.

![Colorizing Objects icon] **Colorizing Objects**

In general, all newly created objects in *Rhino* get the color of the active *Rhino* layer. To differentiate objects from each other by color, select several objects in *Rhino* and click on the **ColorizeObjects** icon to assign different (random) colors to each selected object (Figure 45). To revert to a colorization **by layer**, select objects, open the Properties tab (press **F3**), and in the item labelled *Display Color*, select *Display Color → By Layer*.
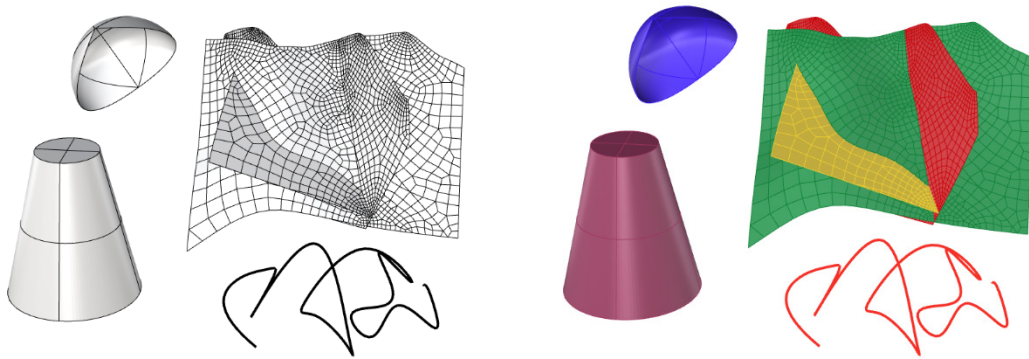

Figure 45: Various objects before and after *ColorizeObjects* is applied.

Group Names Assignment

Groups in *FLAC3D* and *3DEC*

FLAC3D and *3DEC* can associate names with volume elements (zones, blocks) and names or numbers with faces (face groups, joint ids). These named entities make it easier to reference areas of the model for material or boundary condition assignment.

The volume meshers **GVol** and **BlockRanger** deal with groups in a slightly different manner.

- **GVol**, when generating a volume grid, requires a closed set of boundary surface meshes. The inside can be composed of "floating" surfaces, i.e., surfaces that are partially or completely disconnected from other surfaces. Because *Rhino* only operates with surface meshes and not volumes, it is not immediately clear if a set of input surface meshes encloses a watertight volume. For this reason, volumes cannot be directly named; only surfaces can be named.
- **BlockRanger** requires solids as input. There is no ambiguity about what is inside or outside a solid. For this reason, **BlockRanger** can name solids. **BlockRanger** does not allow "floating" surfaces and surfaces in **BlockRanger** do not use names.

GVol Generated Groups – *FLAC3D*

A hypothetical section through a *Griddle* volume mesh generated for *FLAC3D* is shown in Figure 46. After filling the interior of the model domain with zones (elements), *Griddle* assigns an ID to each watertight volume. Figure 46 presents an example of such volume IDs as large white numbers. The IDs are used to generate group names for zones in *FLAC3D*.

Zones
FLAC3D zone groups are always assigned names automatically based on volume ID (as *Rhino* currently does not have capabilities to interact with volumes represented by surface meshes). To distinguish zone groups from other group names, *Griddle* assigns zone groups the prefix "ZG_" and a suffix corresponding to the closed volume ID. For, example, area 1 in Figure 46 can be referenced as "ZG_001" in *FLAC3D*. The "ZG_" groups are stored in slot "SLOT 1".

Zone faces
Zone faces (represented by the thick black and red lines below) are placed in face groups. Face groups can be assigned automatically based on volume IDs or, if faces belong to a named surface, surface name is used. All face groups are placed in slot "SLOT 1".

Griddle creates external face groups with the prefix "EF_" and internal face groups with the prefix "IF_". When surface name is not present (assigned automatically), names of external face groups are appended with the single volume ID to which the faces are attached. Similarly, internal face groups are appended with a combination of two volume IDs representing the two groups on either side of the faces (the larger number is always placed second in the name). If the *Rhino* surface mesh has a name, then this name is always used.

Some surfaces do not fully divide the volume into distinct pieces. Such "floating" surfaces are represented by the red lines in Figure 46. In automatic naming, their face group names contain a duplicate number. For example, faces group "IF_002_002" is designated from the "ZG_002" zone groups on either side of it.

Using Provided Surface Names

If the surface mesh has a name assigned to it in *Rhino* Properties, then this name will be used for *FLAC3D* face group (and prefixed with "EF_" or "IF_"). These names take precedence over the automatically assigned names shown in Figure 46.
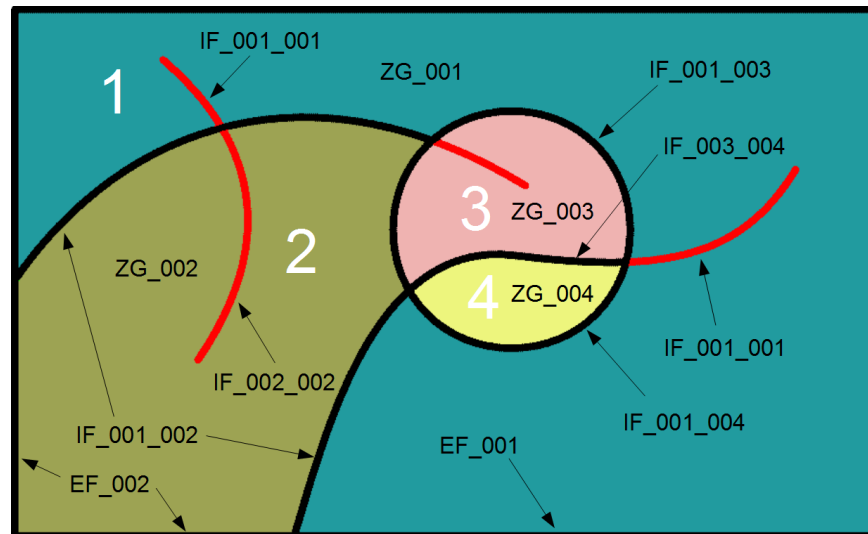

Figure 46: Zone and face groups from a *Griddle*-generated *FLAC3D* grid for automatic group name assignment.

GVol Generated Groups – *3DEC*

Blocks

3DEC block groups are represented by strings containing numeric digits. Block groups start at "001" for 3DEC_*7.x* output and at "10001" for *3DEC 5.2* (due to *3DEC 5.2* specifics) and each new group name is increased by "1". Block groups are assigned in random order. For example, block group names will be "001", "002", … (similar to *FLAC3D* "ZG_…" groups, see Figure 46). All groups are placed in "SLOT 1".

Joint group names

Joint groups are also represented by strings with integers. All external faces and joints within each group of rigid blocks are typically associated with IDs/groups "1" and "2", and they are not output by *GVol*. *GVol* outputs joint groups starting from names "3", "4", "5", … which are given to the surfaces separating various blocks (for *BlockType = Deformable*) or block groups as well as "floating" surfaces (for *BlockType = Rigid*). All groups are placed in "SLOT 1".

Joint groups are arbitrarily assigned if corresponding surface mesh name is not provided (there is no relationship between joint group and the block group names surrounding it). Note that "floating" surface meshes (the red lines in Figure 46) will not appear in *3DEC* deformable output and therefore no names would be assigned to them.

3DEC also allows assigning a separate group name to joints. If a surface mesh has a name assigned to it in *Rhino* Properties, then this name will be output as the joint group name.

BlockRanger-Generated Groups – *FLAC3D*

BlockRanger solids can be named in two ways: through the layer name that the solid belongs to and through the solid Properties Name field (Figure 47). The layer Name is used to generate zone group names for *FLAC3D* in "SLOT 1". The Properties Name is used to generate zone group names for *FLAC3D* in "SLOT 2".

The white text names in Figure 48 and Figure 49 are assumed to be *Rhino* layer names. In Figure 48, the black text prefixed with "ZG_" indicates "SLOT 1" zone groups. The yellow text indicates "SLOT 2" zone groups. "SLOT 2" group names are automatically generated if the Properties Name field is left blank. A *Rhino* layer name cannot be blank. Example internal ("IF_" prefix) and external ("EF_" prefix) face group names are also shown in Figure 48. Note, a face group is not generated for the seam between the two bedrock solids or the seam between the two soil solids shown in Figure 48.
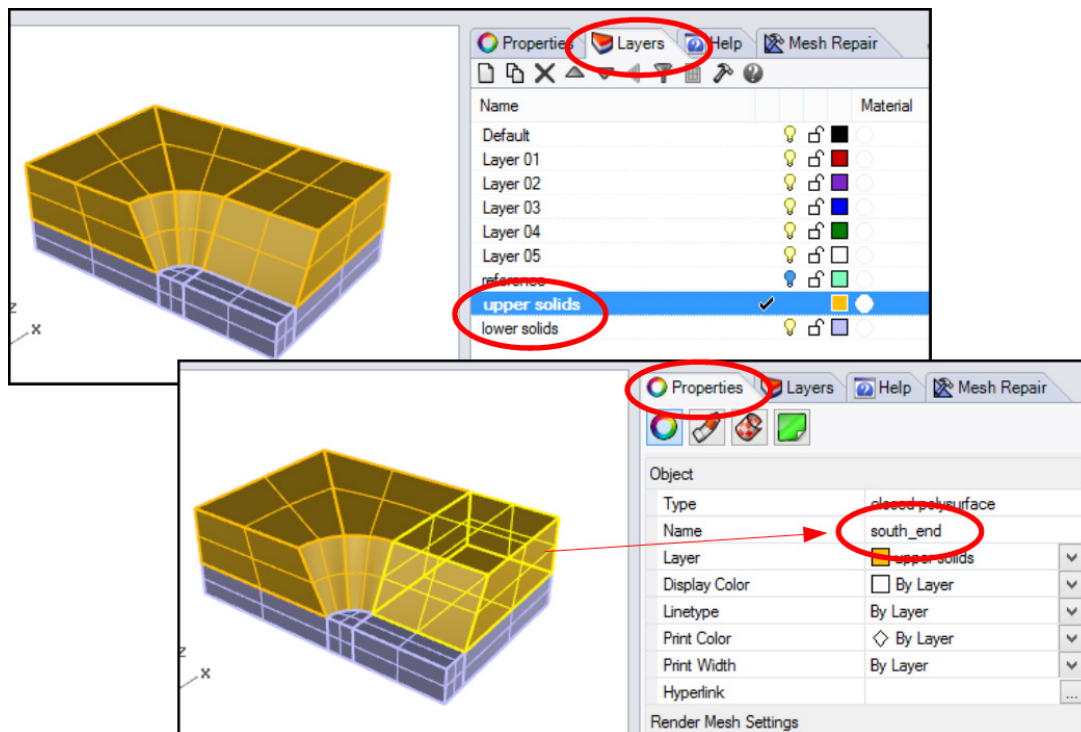

Figure 47: Naming solids by layer (top) and through the Properties Name field (bottom).

BlockRanger Generated Groups – *3DEC*

For *3DEC 5.2*, **BlockRanger** uses the layer name as the *3DEC* block group name (Figure 49). For example, the *Rhino* layer "soil" would result in a *3DEC* block group called "soil". *3DEC 7.x* output uses block numbering similar to *FLAC3D* and does not use layer names. For either version of *3DEC*, all block groups are placed in "SLOT 1". Internal faces which separate different named layers are assigned joint set IDs starting from 2, as in Figure 49.
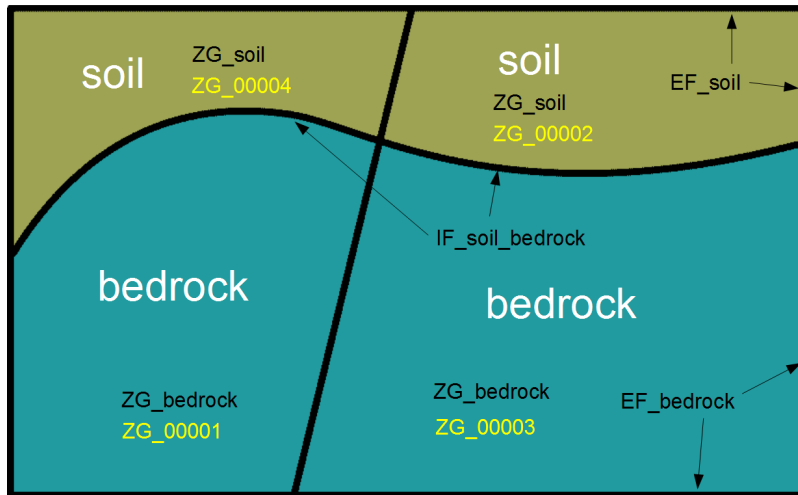
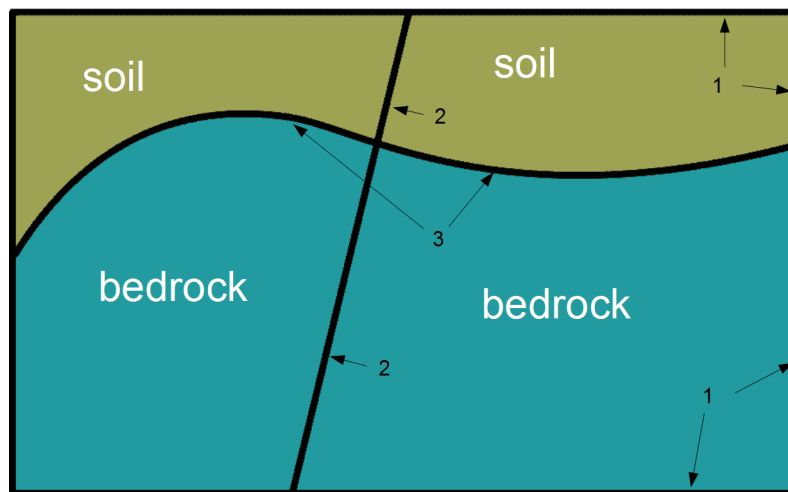**Figure 48: Zone and face groups from a *BlockRanger*-generated *FLAC3D* grid.**

Figure 49: Block groups and joint set IDs from a *BlockRanger* generated *3DEC* model.

Note: *GVol* and *BlockRanger* replace all occurrences of whitespace characters (blanks, tabs, etc.) in name fields with an underscore character "_" to avoid string interpretation problems in *FLAC3D* and *3DEC*. Names are case insensitive, e.g. "Upper_Rock" is treated as the same name as "upper_rock".

Group Names in Other Formats

When outputting in other formats (e.g. *ANSYS*, *NASTRAN*, etc.) no groups are assigned except for <u>*ABAQUS*</u> output in which:

- *GVol* assigns automatically generated names for element groups in watertight volumes in the format: "G10001", "G10002", etc.
- *BlockRanger* assigns names to element groups according to *Rhino* layer names.

References

Itasca Consulting Group Inc. (2019) *FLAC3D — Fast Lagrangian Analysis of Continua in Three Dimensions,* Ver. 7.0. Minneapolis: Itasca.

Itasca Consulting Group Inc. (2020) *3DEC — Three-Dimensional Distinct Element Code,* Ver. 7.0. Minneapolis: Itasca.

Understanding Rhino Tolerances: https://wiki.mcneel.com/Rhino/faqtolerances