# USURP
# Unique Surfaces Using Ranked Polygons
# Users Manual

David A. Boger
Penn State University Applied Research Laboratory

May 27, 2011

## Overview

The purpose of USURP is to enable surface integration on grid assemblies that include overlapping surface grids. The code reads in the overset assembly, removes any overlapping portions, and can integrate the solution on the resulting surface. Users who are familiar with NASA-distributed codes such as OVERFLOW and the Chimera Grid Tools package will recognize USURP as an alternative to FOMOCO, encapsulating the abilities of both MIXSUR and OVERINT into one executable.

USURP is motivated by the "Polymixsur" algorithm presented by Larry Wigton at the 7th Symposium on Overset Composite Grids & Solution Technology at Huntingdon Beach, CA, in 2004, which in turn was motivated by the sometimes troublesome triangulation to form a "zipper grid" in MIXSUR. The present implementation was also motivated by the need to accommodate a broader range of CFD solvers and so was designed to accommodate both cell-centered and vertex-based codes and both structured and unstructured grids.

In the current version, seamless operation is provided for users of several CFD codes, including OVERFLOW, OVER-REL and UNCLE-M (both versions of UNCLE), CFD-SHIP, and DPLR. For users of these codes, no new input data formats or files are needed for use with USURP; instead, USURP will gather the information it needs from the files normally associated with the pertinent CFD code. A generic input capability is available for both structured (PLOT3D) and unstructured (Cobalt) grids. Further details on input data and code usage are provided below.

Output is also environment-dependent but normally consists of a Tecplot data file for visualization of the resulting surface assembly and a list of the surface panel weights computed by USURP. The latter file, called `panel_weights.dat`, can then be read into the CFD solver for incorporation into the native surface integration procedures. Other output files associated with the different CFD environments are described in more detail below.

Because certain parts of the code are still under development, USURP is littered with an array of command line options to provide diagnostic information and to facilitate user control and experimentation. In some cases, the command line options can trigger unstable behavior or gain access to under-developed portions of the code. The default settings are intended to provide the most robust execution, thus minimizing the need for any command line options. No command line options should be needed in the majority of cases.

# Contents


**1 Code Execution** **1**

**2 Input Files** **5**

**3 Output Files** **6**

**4 Post-Processing** **7**

**5 Common Pathologies** **9**

**6 Compiling** **11**

**7 Additional Usage Notes** **15**

7.1 CFD-SHIP . . . . . 15

7.2 DPLR . . . . . 15

7.3 Generic Input File Formats . . . . . 17

7.4 OVER-REL . . . . . 18

7.5 OVERFLOW . . . . . 19

7.6 UNCLE-M . . . . . 21

7.7 Unstructured Grids . . . . . 21

**8 References** **22**

# 1 Code Execution

USURP draws input from both input files and command line flags. The input files vary depending on the CFD environment of the user and are detailed in Sections 2 and 7. The command line flags are detailed here.

The full and current usage for USURP can be obtained from the command line using the command

```
usurp --help
```

While normal usage should not require any command line flags to be specified, many options have been provided to control a variety of code parameters. These options include the following:

`--allow-fringe`: For OVER-REL and UNCLE-M users; specifies whether non-active cells should be included in the initial surface definition. The default value is `.false.`.

`--basis=<panel,patch>`: Choose the basis for polygon ranking as either *panel* or *patch*. When *panel* is chosen, the polygon ranking is based on the original panel area, with smaller panels outranking larger ones. When *patch* is chosen, the polygon ranking is based on the ranking of the patches to which the panels belong. The higher-ranking patch is the one with more panels overlapping the other patch; however, some patch relationships are ignored in order to create an acyclic system. The default value depends on the code environment. For OVERFLOW users, the default is *patch*; for other users, the default is *panel*.

`--check-panels=<p1,p2>`: Explain how two panels interact; i.e. whether or not they are considered to overlap and why. This is intended for debugging purposes and causes the code to stop once the explanation has been given.

`--check-patches=<p1,p2>`: Explain how two patches interact when `--basis` is set to `patch`. This is intended for debugging purposes and reports the initial relative rankings of the two patches and any changes incurred during the linear extension of the graph.

`--color`: Apply a 6-color vertex coloring heuristic to the patch overlap graph for visualization of the patches in Tecplot. If `--tecformat=ascii`, the zone colors are assigned in the zone headers; if `--tecformat=binary`, a separate Tecplot macro (`color.mcr`) is written, and the macro must be played in Tecplot in order to apply the revised coloring.

`--debug`: Display *extremely verbose* debugging information for the R-Tree sort and search, which is definitely *not* recommended.

`--disjoin=<yes,no>`: Specify whether different components interact. The default value depends on the code environment. For OVER-REL users, the default is *no*, which means that differently named surface components in `bc.in` will interact. For OVERFLOW users, the default is also *no*, which is *inconsistent* with the default behavior of MIXSUR. CFD-SHIP users currently have no mechanism for separating components but could consider converting their data into FOMOCO input formats in order to take advantage of this and other features.

`--dplr-input=`*file,version*: Specify the DPLR input file and version. This command line specification of DPLR's input file is available only if USURP has been linked with the DPLR input reader library.

`--dumptree`: Generate a (possibly very large) Tecplot file of the R-Tree, which is definitely *not* recommended.

`--free-memory`: Deallocate memory whenever possible as the code proceeds. This can be time consuming and does not significantly alleviate the memory demands of the code, so it is mainly intended for debugging purposes. In that case, the code will report an error if there is a net number of allocate/deallocate statements during execution, which can help in the prevention of memory leaks.

`--full-surface`: Triangulate the entire surface without attempting to resolve hanging nodes. (Compare this to the `--watertight` option.)

`--help`: Display a brief summary of these command line options and descriptions.

`--ignore-solution`: Ignore any existing solution files and simply process the grid.

`--inflate=`*eps*: Multiply all three dimensions of the axis-aligned panel bounding boxes by *eps* after `--min-factor` and `--min-size` have been applied.

`--input=`*path*: Specify a pathname for the main working directory containing the input files. Default value is the current directory. This option is mainly useful for processing files owned by other users.

`--int=<cp,p>`: For OVERFLOW users; specify whether or not to subtract the freestream pressure during the pressure integration. (The freestream pressure is subtracted by default or if `--int=cp`, unless the solution is specified as incompressible.)

`--keep-weights`: Read the panel weights from `panel_weights.dat` and do *not* re-compute them. For OVERFLOW and OVER-REL users, the purpose is to recompute the forces and moments using previously computed panel weights, which might be desirable if the geometry has been rotated, for example. (Note that the computations performed by USURP depend on the orientation of the geometry relative to the Cartesian axes.)

`--min-factor=`*eps*: Specify the minimum Cartesian dimension allowed for panel bounding boxes to be the product of *eps* and the diagonal length across the initial panel bounding box. Default value is 0.01 (or 1% of the bounding box size). USURP ultimately uses the larger of this dimension and the one resulting from the `--min-size` option.

`--min-size=`*eps*: Specify the minimum Cartesian dimension allowed for panel bounding boxes. Default value is 0 (earlier versions of the code had a default value of 1e-8). This option provides a simple approach for resolving overlaps on flat surfaces where the two grids are separated by small amounts. USURP ultimately uses the larger of this dimension and the one resulting from the `--min-factor` option.

`--never-skip[=tri,clip]`: Never skip triangulation or polygon clipping operations, even if USURP has warned that attempting a particular operation may cause the code to crash. Selecting `--never-skip=clip` will cause the code to repeat any failed polygon clipping operations while skipping any dangerous triangulations. Selecting `--never-skip=tri` will cause the code to attempt all triangulations but not repeat any failed polygon clipping operations. Using `--never-skip=clip,tri` or `--never-skip` alone causes the code to never skip any operation.

`--notree`: Disable the use of the R-Tree in finding overlapping surface panels. This option is provided mainly for debugging purposes in the case where a bug is suspected in the tree-based sort and search. This option is *not* recommended, in part because the alternative simple search

method can be substantially more time-consuming and in part because the R-Tree coding *appears* to be stable and robust.

`--output=`*path*: Specify a pathname where all output files will be written. Default value is the current directory. This option is mainly useful when running the code in a folder where the user does not have write permission.

`--plot-velocity`: include u,v,w in the usurp-surfaces output file.

`--prefix=<RMS,RST,SAV>`: For OVER-REL users; specify the solution file type from which to pull the dependent variables for force and moment integration and for plotting (if `--plot-velocity` has been specified).

`--showcpu`: Display cpu times for each step of execution, with output to STDERR.

`--show-panel=`*index*: Plot the evolution of a specified panel (suitable for subsequent animation using Tecplot). The default is to show no panel. Output is a series of files named `slide*.dat`, which can be animated by modifying and running the Tecplot macro MakeMovie.mcr that is provided along with the USURP source code.

`--tecformat=<ascii,binary>`: Choose *ascii* or *binary* for the Tecplot output file. The *binary* option is only available when the Tecplot I/O library (e.g. `tecio.a`) has been linked during compilation and when the CPP flag `USE_TECPLOT` is defined, both of which can be accomplished by defining the environment variable `TECIO_HOME` to point to the Tecplot library. When the CPP flag `USE_TECPLOT` is defined, the *binary* option is the default.

`--trap[=tri,clip]`: This option can slow the code dramatically and is used only to investigate the cause of code crashes that can occur while clipping polygons in the GPC library or triangulating polygons in the Triangle library. The code outputs a file called `scratch_clipping.dat` just before each clipping operation (if `--trap=clip` is used) and a file called `scratch_triangle.dat` just before each triangulation (if `--trap=tri` is used), and each file is deleted if the operation is successful. In that way, a record of the offending operation is provided in the event of a crash. Specifying `--trap` is the same as specifying `--trap=clip,tri`.

`--ttype=<GPC,Triangle,none>`: Choose *GPC* or *Triangle* for the triangulation method, or choose *none* in order to skip triangulation altogether.

The *Triangle* method is preferred (and is the default option) because it does not add any new nodes during the triangulation; however, the *Triangle* library appears to be much more sensitive to degenerate input sets and can often fail, resulting in segmentation faults. The *GPC* option generates simplistic "tristrips" while adding extra nodes to the grid and is available primarily as an alternative in case the *Triangle* routine fails.

`--use-map`: For OVERFLOW users; create a new `grid.i.triq` file using `usurp.map` (see 7.5)

`--use-priority-pairs`: For OVERFLOW users; over-rule rankings with set priority pairs.

`--verbose[=`*index*`]` (or `-v, -vv`): Provide extra diagnostic information to STDERR. Values for *index* may be 0, 1, or 2, with increasing values providing increasing amounts of diagnostic output. An index of 1 (of `-v`) typically leads to increased screen output, including cpu time spent in each routine (i.e. `--showcpu` is automatically invoked), information regarding the outcome of the patch-based ranking system, and an explicit list of any polygons for which triangulation was skipped. An index of 2 (or `-vv`) further increases the screen output (for example, the reason why a particular triangulation was skipped is given) in addition to providing some diagnostic output files. For example, the file `skipped.dat` is a Tecplot file containing the polygons for which triangulation was skipped.

`--watertight`: Create a watertight triangulated surface by triangulating the entire surface and resolving any hanging nodes introduced by the polygon clipping operation. Triangulation by "Triangle" is required (see the `--ttype` option).

# 2 Input Files

Roughly speaking, USURP requires as input only the surface grids that comprise the integration domain of interest. For convenience, this data is normally drawn from the same input files (grid and boundary condition specification) that are provided to the flow solver. If portions of the surface have

been blanked out by hole-cutting, then the iblank data should also be provided; otherwise, it is assumed that all portions of the surface grid contain valid integration data.

USURP has been written to handle several different CFD working environments (e.g. OVERFLOW, DPLR, CFD-SHIP, etc.). Generally, each working environment is characterized by the existence of a particular file (e.g. `grid.in` for OVERFLOW, `cfd_ship.nml` for CFD-SHIP, etc.). Because DPLR has no such mandatory file names, the input file name (and format) can be specified through the `--dplr-input` command line option. Otherwise, USURP defaults to reading fixed-format DPLR input provided via STDIN when it fails to identify any of the other supported environments. Users should refer to the instructions in Section 7 for instructions specific to each environment.

# 3 Output Files

The primary output from USURP is a "panel weight" associated with each cell face on the integration surface. The panel weight is a real number between 0.0 and 1.0 that multiplies the contribution of that cell face to the overall surface integral.

The `panel_weights.dat` file contains the computed weights of each surface panel for use as input data to the CFD solver. The CFD solver should then be modified to accommodate these weights in its native surface integration routines. The data is stored in an ASCII, formatted file, with the weights written using one record for each grid patch or surface subset, in the order that the patches are encountered on input. For structured grids, within each patch, the data is written using `i,j,k` nested loops (with `i` running fastest) *regardless of the orientation of the surface.*

Several Tecplot files are provided for visualization of the results. For example, the `usurp-surfaces.plt` file provides the mesh definition, panel weights, and a triangulated representation of any partially clipped cell faces.[1] The `color.mcr` file is a Tecplot macro that applies a six-color graph coloring heuristic to the zone mesh colors of `usurp-surfaces.plt`. The macro is only

[1]For the Tecplot files, a file extension of `.plt` denotes that the file contains binary data, whereas the extension is changed to `.dat` if the file contains ASCII data.

provided when the `--color` option is activated and the Tecplot file format is binary. If the Tecplot file format is ASCII, then the color attributes are written directly into the zone headers in the `usurp-surfaces.dat` file. Finally, the `usurp-triangles.plt` file contains the triangulated surface data, provided only when the `--full-surface` or `--watertight` options are specified on the command line.

Section 4 provides recommendations for viewing data within Tecplot. Section 6 provides instructions for compiling USURP to generate binary-formatted Tecplot data files.

Information written to STDOUT consists of basic code diagnostic information. For OVERFLOW users, integration data is written to STDOUT as well; see Section 7.5.

Information written to STDERR consists of fatal error messages as well as cpu time information when the `--showcpu` command line option is used. USURP checks for dozens of run-time errors, which it may report. If such errors occur, report them to dab143@arl.psu.edu.

Some additional output that is specific to certain CFD environments is described further in Section 7.

# 4 Post-Processing

Tecplot is recommended for viewing the results out of USURP because of the flexibility it provides both in plotting a mixture of vertex-based and cell-centered data and in blanking based on a variety and/or combination of criteria.

In Tecplot, load the `usurp-surfaces.dat` or `usurp-surfaces.plt` file. Each zone in the file corresponds to one specified boundary patch, while the last zone contains the triangulated regions. Once the data file is loaded, play the Tecplot macro `setup.mcr` (provided in the USURP source code directory). Frequent users may wish to add this to Tecplot's "Quick Macro" panel. If `setup.mcr` is not available, set the contour variable to *iblank* and set the contour type for all zones to *primary value flood*. Enable blanking for any *primary value* whose *ratio* is *less than 0.999*. Turn the Transparency option

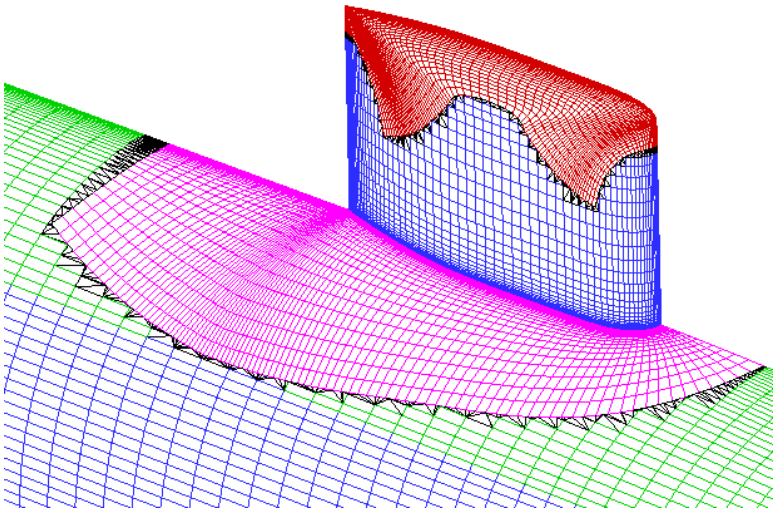


Figure 1: Unique surface mesh resulting from USURP. The surface is colored by iblank to show that only field points contribute to the integration domain

on and the boundaries off.

At this point, the original solid surface should be displayed but without any overlapping regions. (See Figure 1, for example.) If overlapping areas remain, they will appear darker due to the transparency of the surface. The surface should be uniformly colored, showing that all visible panels have an iblank value of 1. In rare cases, a panel having an iblank value less than 0 (fringe) or equal to 101 (orphans) may be partially visible. This would only occur in regions of minimal overlap.

*Very* small gaps will inevitably occur in triangulated regions. This is a function of the method used to triangulate the surface for visualization purposes, which is independent of the determination of the panel weights. As such, it is important only that the gaps be much smaller than the size of the neighboring surface panels.

Errors, if they occur, would manifest themselves as large gaps (implying too much of the surface was removed) or large overlapping regions (implying not enough of the overlapping grid regions were removed). In either case, "large" means on the order of the neighboring surface panel areas (say, 5% or more of

the local panel areas). If the problem appears to be due to separate geometric components in very close proximity (e.g. a hull and appendage with a very small gap between them), the `--disjoin` command line option may be used to prevent this interaction.

Users wishing to eliminate gaps completely from the resulting surface may specify the `--watertight` option on the command line. It should be noted that the logic to make the surface completely watertight in this way is applied only *after* the panel weights are computed and thus has no effect on the integration results.

# 5 Common Pathologies

## Large gaps in the surface

Gaps in the surface may be due to a panel with an iblank value of zero that is not covered by panels with iblank values not equal to zero. The problem can be diagnosed in Tecplot by disabling the blanking and coloring the panels by the contour variable "iblank". As with all scalar variables, the contour attributes should be set to *primary value flood*.

## Overlapping region remains

Overlapping regions may remain when a pair of *apparently* overlapping panels fails to form a *valid* overlapping pair. Most often, this is because the bounding boxes of the two panels fail to overlap. (To find the exact cause for a particular pair of panels, use the `--check-pair` option.) The panel bounding boxes are aligned with the Cartesian axes, so a panel that is perpendicular to one of the Cartesian axes will have zero thickness in one dimension. Therefore, if two panels are intended to lie on the same plane but are in fact lying on parallel planes a small distance apart, their bounding boxes may not overlap. The situation can be remedied using the command line option `--min-size=`*eps* where *eps* specifies the minimum thickness of the panel bounding boxes. Alternatively, set `--min-factor=`*eps*, which sets the minimum thickness to *eps* times the length of the bounding box diagonal.

Naturally, setting the value too large can cause bounding boxes to overlap that should not.

## Small gaps in the surface

Small gaps in the surface (gaps much smaller than the size of the adjoining panels) are to be expected and are a natural consequence of the algorithm. Panels are projected onto a plane prior to undergoing polygon clipping and are rotated back afterwards. The small gaps will be more visible when the four vertices of the adjoining panel are increasingly far from being co-planar.

## Triangles overlapping full panels

Unexpected results are often a result of an interaction between apparently non-overlapping surfaces. Panels will interact when their bounding boxes overlap. Because the bounding boxes are axis-aligned, panels oriented at angles relative to the Cartesian axes may have large bounding boxes. Panels whose bounding boxes thus overlap may clip one another if the dot product of their surface normals is greater than zero. Therefore, a tapered control fin may interact with the body from which it extends since the taper causes the dot products of the surface normals to be slightly greater than zero. One way to identify the cause of such problems in Tecplot is to color the panels by the contour variable *ipan*, which can help to identify the source of the intruding panel. Alternatively, the Tecplot probe tool can be used to identify the panel numbers and the `--check-pair` option can then be used to discover the cause of the problem. The problem can be corrected by separating the surfaces into different components and using the command line option `--disjoin` or `--disjoin=yes`.

## Segmentation faults

If using Intel Fortran compilers, be sure to set "ulimit -s unlimited." Intel compilers starting with version 8 use more stack for storage of temporary arrays, requiring care that enough stack is available.

If the segmentation fault occurs during the GPC clipping (after the R-tree is built and while the code is within the GPC routines), then try adjusting GPC_EPSILON (in `gpc.h`), either by raising it or lowering it by an order of magnitude or two. Increasing the value encourages merging of points, which helps prevent lots of tiny elements from being left over in the final surface. The GPC library was written with the tolerance very small, however, and increasing it might be causing the robustness problem. Setting `--trap=clip` can isolate the polygon clipping operation that is causing the failure by writing a file called `scratch_clipping.dat` prior to the Boolean difference operation for each pair, but note that this option can dramatically increase the time it takes to run the code.

If the segmentation fault occurs during the triangulation when Triangle is used, then there is a good chance that the polygon data being passed to Triangle is degenerate or does not otherwise meet Triangle's demanding standards. Setting `--trap=tri` can isolate the problematic polygon by writing a file called `scratch_triangle.dat` prior to triangulating each polygon, but note that this option can dramatically increase the time it takes to run the code. The GPC tristrips seem to be more lenient in this respect, so switch to `--ttype=GPC` if possible or else modify the `GarbageCollector` routine to clean up the polygon. Specifying `--ttype=none` will avoid the triangulation altogether.

If segmentation faults are proving more of a problem than run time, then it is probably worthwhile to always compile the code with the debugging flags so that when a problem does arise it can be tracked down. Often, a problem that arises while the code is optimized may disappear again when the code is recompiled with the debugging options on.

# 6 Compiling

To build the USURP executable, change to the `src/` directory and issue the command

```
make <machine> [CMD=<cmd>]
```

where `<machine>` is any one of the machine-dependent targets defined in `Make.sys`, which is included into Makefile. For example, "`make intel8`" compiles USURP using the Intel compiler with flags set to use big endian format for input and output. Simply typing "`make`" with no target will list all of the available pre-defined targets. Note that the code is always compiled in double precision, even for single-precision input/output.

Targets in `Make.sys` have been previously defined for the Portland Group compilers (pgf90 and pgcc under Linux), XL Fortran for AIX (xlf90 and xlc under AIX), Sun f90, Intel ifort, SGI and Lahey/Fujitsu Fortran 95 (lf95), and g95. Other targets exist in `Make.sys` but are commented out because they have not been tested with USURP.

`makemake.pl` is a Perl script that can be used to regenerate the Makefile dependencies if modifications are made to the code. Therefore, changes should be made to `makemake.pl` or `Make.sys`, not to `Makefile` itself.

USURP is written using (mostly) standard Fortran 90 and is linked to the General Polygon Clipping (GPC) and Triangle libraries, which are written in C. The code should be compiled with `-DMCLOCK` when the intrinsic function `mclock` is available, or with `-DCPU_TIME` when the intrinsic subroutine `cpu_time` is available. Byte swapping options are normally specified through the compiler options (such as `-Mbyteswapio` under Portland Group's pgf90) but can be specified through the use of `-DCONVERT_BIG_ENDIAN` when the `convert` attribute of `open` is available.

If SUGGAR DCI files are used to provide the iblank values for unstructured grids, then USURP must be linked with the DiRT library. In that case, the environment variables `DIRTLIB_DIR` and `P3DLIB_DIR` must be defined to point to the *directories* containing `libdirt.a` and `libp3d.a` respectively.

If binary Tecplot output files are desired, the `TECIO_HOME` environment variable must be defined to point to the Tecplot library *file* (e.g. `tecio.a`) distributed with Tecplot. The non-standard `%VAL` intrinsic that is used in the code is associated with the binary Tecplot output. If the command is a problem, remove the binary Tecplot output calls from the compilation (e.g. `unsetenv TECIO_HOME`).

If compilation results in the error

```
undefined reference to __gxx_personality_v0
```

then add `-lstdc++` and/or `-lsupc++` to the `TECIO_HOME` definition or remove Tecplot from the compilation (see the preceding paragraph). To add the C libraries on the command line, use

```
make <machine> LIBS="-lstdc++ -lsupc++"
```

If the `TECIO_HOME` variable has been defined (see above), then it may be easier to change the `TECIO_HOME` definition to include the C++ libraries, e.g.

```
setenv TECIO_HOME "/path/tecplot/tecio.a -lstdc++ -lsupc++"
```

Another alternative is to statically link the C library `ctypeb` to the Tecplot library file by issuing the following commands:[2]

```
cp $TECIO_HOME .
gcc -c ctype.c
ar -qv tecio.a ctype.o
setenv TECIO_HOME ./tecio.a
```

A copy of `ctype.c` has been provided in the subdirectory named `extra/`.

USURP relies on the General Polygon Clipping Library (Version 2.32) by Alan Murta for polygon Boolean operations and triangulation. The GPC library is free for non-commercial use. Anyone wishing to use the GPC library in support of a commercial product should email gpc@cs.man.ac.uk. (A full copyright notice is included in `gpc.c` in the `GPC2.32` subdirectory in `extra/`.)

The GPC library uses a tolerance for merging points, which is set in `gpc.h`. The original code uses `DBL_EPSILON` for this tolerance, which can cause very small triangles to remain in the final surface definition. Raising the tolerance to 1E-06 eliminates this but can lead to segmentation faults (particularly ahead of `merge_right` and `merge_left`) in the `gpc_polygon_clip` module

[2]Thanks to Ben Kirk of NASA-JSC for providing this suggestion.

in `gpc.c`. Some error handling has been added to skip these polygons in these cases and issue a warning message, but if such warning messages persist, the tolerance in `gpc.h` should be addressed. To avoid crashes, Murta recommends keeping the tolerance as small as possible.

USURP also relies on Triangle (Version 1.6) by Jonathan Shewchuk for triangulation. Please note that although Triangle is freely available, it is copyrighted by the Jonathan Shewchuk and may not be sold or included in commercial products without a license. (A full copyright notice is included in the `README` file of the `TRIANGLE1.6` subdirectory in `extra/`.) Specific instructions for compiling and using the Triangle library are in the `TRIANGLE1.6` subdirectory.

The Triangle library, which is written in C, has extensive documentation which has been copied to the `TRIANGLE1.6` subdirectory. Included in the documentation is advice on compiling the library, which recommends defining `REDUCED` and `CDT_ONLY` to reduce the size of the library object file, `SELF_CHECK` to have the routine report diagnostic error messages, and `LINUX` for proper behavior on Linux machines. These definitions are included in `CFLAGS` in `Make.sys`.

The compilation of Triangle on Linux machines (where the `LINUX` CPP flag must be defined) requires the use of the `fpu_control.h` header file, which in turn requires support for in-line assembly commands. When Portland Group compilers are used, the in-line assembly feature requires version 6.1 of pgcc or later to compile. If this version is not available, gcc can be used instead. On non-Linux machines, the `LINUX` flag should not be defined. The `README` file in the `extra/TRIANGLE1.6` directory provides more information on how to compile the Triangle library.

If FXDR-formatted grid files are used in conjunction with a DPLR input file, then in order to use the XDR binary format, USURP must be linked to the FXDR library during compilation. The user is responsible for downloading and compiling the FXDR library.[3] In that case, the environment variable FXDR_HOME must be defined to point to the FXDR archive (.a) *file*.

DPLR users can also choose to link USURP to the DPLR input file reader routines directly, which then allows USURP to read any version of both

[3]The FXDR library was written by David W. Pierce and can be obtained from http://meteora.ucsd.edu/~pierce/fxdr_home_page.html.

fixed- and free-formatted DPLR input files. In order to link with the DPLR library, the DPLR_HOME environment variable must be defined to point to the DPLR source code, such that the DPLR_HOME/include DPLR_HOME/dpinputs directories exist.

A configure script has been added to the main directory. In theory, it can be used instead of makemake.pl to generate the Makefile, but it has not yet been adequately tested. Adventurous users are welcome to try the commands

```
configure
make
```

from the main directory and relay their experience.

# 7 Additional Usage Notes

USURP should be able to draw the information it needs from existing data files already provided to the flow solver. As such, the input files and usage notes vary to some degree depending on the associated flow solver. The sections below provide details specific to each code.

## 7.1 CFD-SHIP

The presence of a file named cfd_ship.nml signals USURP to read input data associated with CFD-SHIP.

USURP draws the name of the grid (*grfd) file and boundary condition (*bcs) file from cfd_ship.nml. For overset cases, the Pegasus XINTOUT file is also examined for the iblank values of the surface grid points.

## 7.2 DPLR

Because DPLR has no mandatory file names, USURP provides two mechanisms for accessing the input files; the filename and format are either provided

to DPLR via the `--dplr-input` command line option, or else DPLR defaults to reading input associated with DPLR from STDIN when it fails to identify any of the other supported environments.

The `--dplr-input` command line option allows the user to specify the DPLR input file name and version. This is the recommended method for DPLR users to specify their input because USURP can then use the DPLR input modules to read any version for both fixed- and free-formatted DPLR input files. However, to use this option, USURP must be linked with the DPLR reader modules when USURP is compiled; see Section 6. In addition, it is recommended to specify the `DPLR_INPUT_PATH` environment variable to point to the directory where the chemistry files are located in order to satisfy the DPLR reader modules.

If the DPLR reader modules are not linked to USURP, then the flow solver input file for DPLR may be provided to USURP by redirection of the standard input. The basic usage in this case is simply

```
usurp < filename
```

where `filename` is the name of the DPLR flow solver input file. This method is suitable for fixed-formatted DPLR input files only, and for DPLR version 4.03 and earlier.

Required input to USURP for DPLR users consists of the DPLR flow solver input file and the associated grid file (i.e. the grid file identified in the flow solver input file). USURP supports all three formats for the parallel archive grid file that DPLR supports, namely native unformatted (type 1), XDR binary (type 11), and ASCII formatted (type 21).[4]

Most USURP command line options are available to DPLR users. Note, however, that USURP serves only as a pre-processor to POSTFLOW. That is, USURP will generate the panel weights (`panel_weights.dat`) for use in surface integrations conducted within POSTFLOW, but no stand-alone integration of the dependent variables is conducted within USURP.

The DPLR user's guide provides more information on running POSTFLOW

[4]See Section 6 for additional compilation instructions required to read XDR-formatted grid files.

for overset grids and for merging POSTFLOW output with USURP output for generating surface plots.

## 7.3 Generic Input File Formats

Structured grid data associated with codes other than those explicitly mentioned in this document can be converted into a generic format for use with USURP. In such cases, the presence of a file named `generic.bc` signals USURP to read the generic input data. The following files are then required to be in the working directory: `generic.bc`, `generic.grd`, and in some cases, `generic.ib`. These files are described below.

The `generic.bc` file is an ASCII file containing a list of the solid surfaces. Each line takes the form

```
block, i1, i2, j1, j2, k1, k2
```

where `block` is the block number, `i1,i2` is the grid point range of the surface in the i-direction, `j1,j2` is the grid point range in the j-direction, and `k1,k2` is the grid point range in the k-direction. Grid point ranges are provided using a vertex-based numbering system. For example, a j-min surface in the 4th block of a grid system might be specified as

```
4,  1,33,  1,1,  1,17
```

Currently, there is no provision to label components (i.e. there is no provision to group surfaces) under the generic input file format, which effects only the use of the `--disjoin` command line option.

The `generic.grd` file must contain the grid in ASCII, PLOT3D, multiblock format, which is specified in detail below. As the grid is in ASCII format, the grid coordinates may be expressed using single or double precision. The file should be written using commands equivalent to these:

```
open(unit=3,file=generic.grd,form=formatted)
```

```
write(3,*)nb
write(3,*)(ni(n),nj(n),nk(n),n=1,nb)
do n = 1,nb
write(3,*)(((x(i,j,k,n),i=1,ni(n)),j=1,nj(n)),k=1,nk(n)), &
          (((y(i,j,k,n),i=1,ni(n)),j=1,nj(n)),k=1,nk(n)), &
          (((z(i,j,k,n),i=1,ni(n)),j=1,nj(n)),k=1,nk(n))
end do
close(3)
```

The `generic.ib` file provides the cell-centered iblank values for each computational cell. If the `generic.ib` file exists, USURP interprets the iblank values to have the following meanings:

iblank = 1: active or field cells
iblank < 0: fringe or recipient cells
iblank = 101: orphan cells
iblank = 0: hole or out cells

Note that all iblank values less than zero are considered to be equivalent. If `generic.ib` does not exist, the iblank value is assumed to be equal to 1 for all cells.

The `generic.ib` file should be written using commands equivalent to these:

```
open(unit=3,file=generic.ib,form=formatted)
do n = 1,nb
  write(3,*)(((ib(i,j,k,n),i=2,ni(n)),j=2,nj(n)),k=2,nk(n))
end do
close(3)
```

Note that the `ib` values are read as integers and that the index range (e.g. `2:ni(n)`) includes only the interior cells (one less than the number of vertices in each direction).

## 7.4 OVER-REL

The presence of a file named `over_rel.nml` signals USURP to read input data associated with OVER-REL.

Case information is taken from `over_rel.nml`, including the path to the Suggar DCI file when `overset` is set to `true`. Boundary condition information is drawn from `bc.in`, and the surface grid information associated with solid surface patches is drawn from the grid (`G*`) files in the `IO_FILES` subdirectory.

If solution (e.g. `RST*`) files are present, then surface integration is performed to calculate the hydrodynamic forces and moments. In that case, the integration also relies on the eddy viscosity (`MUT*`) files (for best results), and distance (`D*`) files (if `IWFN=1`). The results of the integration are output to `usurp_data.txt`, an Excel file containing pressure, viscous, and total forces and moments on each component. Moments are measured about the point (`xcg`,`ycg`,`zcg`), which is obtained from `over_rel.nml`.

Data in `usurp_data.txt` (output by USURP) should generally agree with the data in `component_data.txt` (output by OVER-REL), provided, of course, that the `panel_weights.dat` file was generated during pre-processing and made available to OVER-REL. In addition, USURP does not incorporate any grid velocity or dynamic transforms into its integration routine, so agreement can only be expected in static cases having zero grid velocity.

## 7.5 OVERFLOW

The presence of a file named `grid.in` signals USURP to read input data associated with OVERFLOW (or more specifically, FOMOCO).

Two modes of operation are available to OVERFLOW users: an "OVERFLOW" mode and a "stand-alone" mode.

The "stand-alone mode" for USURP is accomplished by using USURP as a *post*-processing code. In this instance, USURP will read the grid file (which must be named `grid.in`) as well as input from STDIN. The input data stream should follow the same conventions as the standard input for MIXSUR. If the solution file (`q.save`) exists, USURP will read it and use it to

conduct the force and moment integration; otherwise, it simply generates the panel weights.

The pressure, viscous, and total forces and moments for each integration subset and each integration surface is written to STDOUT, consistent with output generated by OVERINT. As with OVERINT, the origin for the moments is specified through STDIN in the MIXSUR input format.

The "OVERFLOW mode" for USURP is accomplished by using USURP as a *pre*-processing code to generate the panel weights file (`panel_weights.dat`), as well as the MIXSUR parameter file (`mixsur.fmp`) and an integration iblank file (`grid.ibi`). OVERFLOW will recognize and read the panel weights file and apply the panel weights to its native surface force and moment integration. Note that the integration iblank file presently sets all iblank values to 1; any regions of the grid that should be blanked out by the iblank values should have panel weights of zero set automatically by USURP.

The grid file (`grid.in`) must be unformatted but can be single- or double-precision, and multiple- or single-grid PLOT3D format. The grid file can include iblank data, but if it does not, then iblank is assumed to be equal to 1 for all points. The flow file (`q.save`) can be formatted as a PLOT3D q-file or OVERFLOW solution file; however, it must otherwise have the same format as the grid file (`grid.in`) as far as whether the data is in single or double precision and whether the file is in multiple- or single-grid PLOT3D format.

Priority pair data is ignored by USURP by default but can be taken into consideration by adding the `--use-priority-pairs` option to the command line.

Other output files include `mixsur.fmp`, `grid.i.tri`, and `grid.ibi`. The `mixsur.fmp` file is an ASCII file containing a summary of the MIXSUR-style input, including integration subsets, surfaces, and components, and reference conditions. This file is required for input to OVERFLOW when using USURP in "OVERFLOW mode." The `grid.i.tri` file is an ASCII file containing the triangulated surface representation, provided only when the `--full-surface` or `--watertight` options are included on the command line. If the solution file is present, the file will instead be named `grid.i.triq` and include the conservative variable data. Finally, `grid.ibi` is an unformatted file containing the grid and iblank information for the integration subsets. This file is required for input to OVERFLOW when using USURP

in "OVERFLOW mode." The file is either single- or double-precision, consistent with the precision of `grid.in`. Presently, the iblank values are set to one everywhere since it is assumed that the USURP `panel_weights.dat` file is all that is needed to carry out the surface integration correctly.

The command line option `--use-map` can be used to generate a `grid.i.triq` file given a `grid.i.tri` file and a matching `usurp.map` file without having to repeat the polygon clipping or triangulation steps in USURP.[5] To accomplish this, first run the command

```
usurp --full-surface --ignore-solution < mixsur.i
```

in order to generate the `grid.i.tri` file and matching `usurp.map` file. (The `--watertight` option can be specified in place of `--full-surface`.) Then, given a new `q.save` file, issue the command

```
usurp --use-map < mixsur.i
```

in order to generate the corresponding `grid.i.triq` file.

## 7.6 UNCLE-M

The presence of a file named `uncle.inp.bcs` or `uncle.inp` signals USURP to read input data associated with UNCLE-M.

Boundary patch information is taken from the input file, and the surface grid information associated with solid surface patches is drawn from the grid (`uncle.grd*`) files in the `GRIDS` subdirectory. The `uncle.lst` or `uncle.inp` file is parsed for the path to the Suggar DCI file if `overset` is set to `true`.

## 7.7 Unstructured Grids

USURP supports the use of unstructured grids specified in the Cobalt format. This execution mode is triggered by the presence of the file `cobalt.inp`

[5] beginning with USURP version 2.33

(containing the grid in Cobalt format) and also requires the file `cobalt.bc` containing the boundary specification. The cell-centered iblank information, if needed, must be specified using the Suggar DCI format and currently must be contained in a file named `nphase.dci`.

# 8 References


Boger, David A., and Dreyer, James J., "Prediction of Hydrodynamic Forces and Moments for Underwater Vehicles Using Overset Grids," AIAA-2006-1148, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 9-12 January 2006.

Chan, William M., and Buning, Pieter G., "User's Manual for FOMOCO Utilities – Force and Moment Computation Tools for Overset Grids," NASA Technical Memorandum 110408, July 1996.

Guttman, Antonin, "R-Trees: A Dynamic Index Structure for Spatial Searching," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 47-57, 1984.

Murta, Alan, "A General Polygon Clipping Library," Version 2.32, Advanced Interfaces Group, Department of Computer Science, University of Manchester [6]

Shewchuk, Jonathan Richard, "Triangle," Version 1.6, Computer Science Division, University of California at Berkeley [7]

Wigton, Laurence B., "Polymixsur – Boeing's Replacement for Mixsur," 7th Symposium on Overset Composite Grids & Solution Technology, Huntingdon Beach, CA, October 5-7, 2004.


[6] http://www.cs.man.ac.uk/aig/staff/alan/software/gpc.html
[7] http://www.cs.cmu.edu/~quake/triangle.html