

readgeometry

November 24, 2009
13:27

Contents

1	External Geometry Interface	1
2	INDEX	61

1 External Geometry Interface

This program reads a geometry specification from an external file, and transforms it into the form required by DEGAS 2. Namely, the hierarchy of geometry-related objects in DEGAS 2 is, from the lowest level to the highest level is:

1. surface,
2. cell,
3. polygon,
4. zone.

Individual cells are composed of surfaces, as discussed in the documentation for the internal geometry (e.g., see *geometry.web*). The next level up is a polygon. The code automatically breaks up (non-convex) polygons into cells. Finally, a zone may consist of one or more polygons; properties are constant across a zone. For example, a single zone might be used to represent the vacuum region around the plasma which is comprised of several (possibly disconnected) polygons. Or, a plasma flux surface on which density and temperature are constant might be a single zone. Note that polygons are used only within external interfaces such as this code. Cells and surfaces are essentially used only by the code itself. Zones are used primarily for the specification of input and output data.

The interfaces between the plasma (or vacuum) and a solid surface (or exit) are represented by sectors, which are established for diagnostic purposes. In general, a sector is defined by a surface and a zone. This code makes an attempt to identify and label these sectors automatically.

Note first that the input files for `readgeometry` are not listed in `degas2.in`. Although this might be viewed as an omission, it is consistent in that `readgeometry` is, in principle, one of many different means of generating DEGAS 2 geometries and, thus, is not as fundamental as the other entries in `degas2.in`. At present, the only alternative approach is `boxgen`, but other options may soon be available.

Instead, the command line for the execution of `readgeometry` specifies the name of the main `readgeometry` input file. This file will contain a pointer to the second file which will provide externally generated, detailed information about the geometry. Presently, this file can be one of:

1. An input file for the original DEGAS code,
2. A geometry and plasma description (in a specific format) taken from UEDGE,
3. A `.elements` file of the sort generated by SONNET.

Note: because the interface between UEDGE and definegeometry2d has continued to develop while the one with readgeometry has remained stagnant, users are strongly recommended to consider using definegeometry2d.

The details of these files will eventually be described elsewhere (e.g., in the DEGAS manual). Here, we describe the shorter, first file which provides the additional information required to transform the externally generated geometry data into a consistent DEGAS 2 geometry specification.

Because the UEDGE and SONNET geometries are relatively specific, the additional assumptions required to complete the DEGAS 2 geometry can mostly be hardwired into the `readgeometry` code. In this case, the main `readgeometry` input file is very simple.

However, the flexibility of the old DEGAS code makes it difficult to set up a generally applicable means of converting the geometry into DEGAS 2 form. Most of the keywords in the main `readgeometry` input file are concerned with manipulating the DEGAS geometry data.

Each line in the `readgeometry` input file is of the form:

keyword *arguments*

Some keywords have no arguments, however.

Here are the possible keywords and their arguments; keywords will be bold-faced, arguments will be italicized. The `symmetry` and one of the three keywords following it in the first list below must appear at the top of the `readgeometry` input file. As in the other DEGAS 2 input files, lines beginning with a `#` sign in column 1 are ignored (comments) as are blank lines.

symmetry *sym* describes the symmetry of the geometry with `sym = plane` or `cylindrical` (“toroidal symmetry”). This must be the first keyword in the file.

degasfile *filename* provides the name of a DEGAS input file.

uedgefile *filename* provides the name of a UEDGE-generated geometry / plasma file.

sonnetfile *filename* provides the name of a SONNET `.elements` file.

The keywords which apply only in the **degasfile** case are:

minx *min* tells the code to exclude grid indices for which the `gridx` value is less than *min*.

minz *min* tells the code to exclude grid indices for which the `gridz` value is less than *min*.

mindensity *min* excludes grid indices which corresponding to cells with plasma density less than the *min*.

grid *j1 j2 j3 j4 sense* is used to transform the DEGAS `gridx` and `gridz` arrays into DEGAS 2 zones using the following guidelines:

1. The vertical boundary (or horizontal zone, i.e., the first index of `gridx` and `gridz`) limits are *j1* and *j2*. The horizontal boundary (or vertical zone, the second index) limits are *j3* and *j4*.
2. For each pair of integer coordinates in this implied list, a separate DEGAS 2 zone of type `plasma` will be generated. Note that this includes points at $[j2] + 1$ and $[j4] + 1$ (hence, these integers would be more properly associated with DEGAS’s zone indices).

3. Zones can be excluded from this list by using **minx**, **minz**, or **mindensity** keywords in this input file. The code will also ignore zones in which the DEGAS arrays **kzone1** or **kzone2** are less than or equal to zero.
4. **sense** is optional. Allowed values are **sense** = 1 (default) and 2. A value of 1 indicates that a clockwise trip around a single cell requires incrementing the first index of **gridx** and **gridz** initially, then incrementing the second index to get the third point, etc. Likewise, a value of 2 implies that the index must be incremented first to make this clockwise trip.

new_zone type starts a new zone; **type** is a string that indicates the type of the zone. It can be **plasma**, **vacuum** or **solid**.

new_polygon starts a new polygon which will be added to the zone begun by the last **new_zone** keyword. This line should be followed by one or more lines which specify the polygon coordinates.

wall wall start stop direction takes points from wall number **wall** of the DEGAS **xwall** and **zwall** arrays and adds them to the current polygon.

1. **start** and **stop** can be replaced by a “*”, which will add the whole wall.
2. **direction** is optional. Use a positive value (default is +1) if **start** is less than **stop**; use a negative value to traverse the wall in the opposite direction.

outer i j k adds the *i*th *j*th and *k*th points of the bounding rectangle of the problem (minimum, maximum *x* and *z* used to define DEGAS 2’s “universal cell”) to the current polygon. The numbering of the corners is clockwise starting from 0 at x_{\min} , z_{\min} . There can be any number of arguments, although typical usage will be with three.

edge j1 j2 j3 j4 takes an “edge” (i.e., a section of a horizontal or vertical boundary) from the DEGAS **gridx**, **gridz** arrays and adds it to the current polygon. The arguments indicate the same things as in grid except that here we must have either $j1 = j2$ or $j3 = j4$ in order for the edge to lie on a single boundary.

define_polygon terminates the definition of the current polygon begun by the previous **new_polygon** and adds it to the current zone.

Keywords which apply only in the UEDGE and Sonnet cases (the only difference between the two is that double-null option is currently not available; the reason has been forgotten!) are:

mesh rmin rmax zmin zmax defines the plasma zones based on the UEDGE or Sonnet mesh. The input parameters are used to specify the universal cell. The corresponding rectangle, therefore, must completely enclose the mesh coordinates.

null_type j can be $j = 1$ (single-null) or 2 (double-null).

wallfile filename specifies a file with two wall segments which will be used to set up vacuum regions around the plasma zones established via a UEDGE file.

These keywords are used to label solid and exit polygons. They may be used with any of the three input file types:

stratum j sets the stratum label to be *j*. Polygons defined subsequently will be labeled with this stratum number until the label is changed again via this keyword. For DEGAS input files, **readgeometry** will use the DEGAS wall number as the stratum label, unless overridden by explicit use of this keyword.

material *symbol* sets the label for the current material to be that represented by *symbol* (an entry in the `materials.input` file. Polygons defined subsequently will be made of this material until the label is changed again via this keyword. For DEGAS input files, `readgeometry` will attempt to extract the material from the `kmat` array, unless overridden by explicit use of this keyword. In other words, if the user wishes to set the current material explicitly, the **material** keyword should come *after* any **wall** keywords used in defining the polygon (and before **define_polygon**).

temperature *value* assigns the current temperature to *value* kelvin. This is used along with the material and stratum labels in defining the material surfaces (targets and walls) in the problem. If not set, a default temperature of 300 K is used. For DEGAS input files, `readgeometry` will attempt to extract the material from the `twall` array, unless overridden by explicit use of this keyword.

recyc_coef *value* assigns the current recycling fraction, handled in the same way as the current material and temperature, to *value*, which must be ≥ 0 and ≤ 1 . The absorbed fraction is $1 - \text{value}$. The default recycling coefficient is 1. While the DEGAS variable `frabsorb` specifies the fraction of the not-reflected atoms which are absorbed; i.e., the overall absorption fraction depends on the local reflection coefficient. Hence, we have not attempted to relate the DEGAS 2 recycling coefficient to `frabsorb`.

end terminates the input to `readgeometry`.

A typical main `readgeometry` input file for reading DEGAS input will have a structure similar to:

```

symmetry ...

degasfile ...

minx ...
minz ...
mindensity ...
grid

new_zone ...
  new_polygon
    wall ...
    outer ...
    edge ...
    material ...
    stratum ...
  define_polygon

  new_polygon
  .
  .
  .

new_zone
.
.
.

end

```

The input files for the UEDGE cases are usually much simpler. This is possible since the UEDGE case is more specific and much of the topological information can be hard-wired into the code:

```
symmetry ...  
  
uedgefile ...  
  
null_type ...  
wall_file ...  
material ...  
mesh ...  
end
```

See the Examples section in the manual for additional examples of `readgeometry` input files.

```
$Id: readgeometry.web,v 1.46 2007/06/19 17:43:15 dstotler Exp $
```

```
"readgeometry.f" 1 ≡  
@m FILE 'readgeometry.web'
```

The unnamed module.

```
"readgeometry.f" 1.1 ≡  
@f namelist integer
```

⟨Functions and Subroutines 1.2⟩

The main program

```
"readgeometry.f" 1.2 ≡
@m mem_delta 10
@m xz_index 2
@m num_points 200

/* Code specific to a few geometries, e.g., for detector setup, appears in this file. Unless you
know for a fact that your geometry corresponds to one of the other types, set GEOMETRY to
GENERAL. */

@m GENERAL 0
@m UEDGE_RECT_SLAB 1 // Marv's single-null slab used in the EIRENE benchmark.
@m BOX 2 // A 1 by 1 box, e.g., from boxgen.
@m WISING_CMOD 3 // Fred Wising's UEDGE simulation of C-Mod.
@m DEGAS_CMOD 4

@m GEOMETRY GENERAL // Pick one of the above.

@m next_surface #:0 // Statement label

@m increment_num_polygon num_polygon++
var_realloca(polygon_x)
var_realloca(polygon_temperature)
var_realloca(polygon_recyc_coef)
var_realloca(polygon_points)
var_realloca(polygon_zone)
var_realloca(polygon_material)
var_realloca(polygon_stratum)
var_realloca(polygon_segment)

⟨Functions and Subroutines 1.2⟩ ≡
program readgeometry
  implicit_none_f77
  implicit_none_f90
  character*FILELEN filename

  call readfilenames

  call command_arg(1, filename)
  call nc_read_materials
  call readgeomfile(filename)

  stop
end
```

See also sections 1.3, 1.12, 1.18, 1.19, 1.20, and 1.21.

This code is used in section 1.1.

The main routine.

(Functions and Subroutines 1.2) +=

```

subroutine readgeomfile(filename)

  define_dimen(poly_ind, num_polygon)
  define_dimen(xz_ind, xz_index)
  define_dimen(points_ind0, 0, num_points - 1)
  define_dimen(points_ind, num_points - 1)
  define_varp(polygon_x, FLOAT, xz_ind, points_ind0, poly_ind)
  define_varp(polygon_temperature, FLOAT, poly_ind)
  define_varp(polygon_recyc_coef, FLOAT, poly_ind)
  define_varp(polygon_points, INT, poly_ind)
  define_varp(polygon_zone, INT, poly_ind)
  define_varp(polygon_stratum, INT, poly_ind)
  define_varp(polygon_material, INT, poly_ind)
  define_varp(polygon_segment, INT, points_ind, poly_ind)

  implicit_none_f77
  gi_common
  sc_common
  zn_common
  ma_common
  implicit_none_f90
  integer mx, my
  parameter (mx = 120, my = 100)
  character*FILELEN filename, datafilename, wallfilename
  logical uedge, degas, sonnet, single_null, double_null
  integer length, p, b, e // Local
  integer datafile_diskin, wallfile_diskin, zone, n, wall_no, wall_start, wall_stop, wall_direction, i, j,
    grid_startv, grid_stopv, j1, j2, grid_starth, grid_stoph, steph, stepv, h, v, num, nx, ny, nu,
    ixpt1, ixpt2, iysptrx, symmetry, ix_mirror, null_type, grid_sense, segment, segment1
  integer zonearray0:0, facearray0:1
  @#if (GEOMETRY == UEDGE.RECT.SLAB V GEOMETRY == DEGAS.CMOD)
    integer var, tab_index, spacing
    real var_min, var_max, mult
    integer grp_sectors1 mx
  @#endif

  character*LINELEN line, keyword, new_zone_type
  logical complete, vacuum
  real xmin, xmax, ymin, ymax, zmin, zmax, vol, vola, vol_test
  real mindensity, minx, minz, current_temperature, current_recyc_coef
  real wall_recyc, target_recyc, pfr_recyc
  real rmmx, my, 0:4, zmmx, my, 0:4, x_test2, 0:4

  vc_decl(min_corner)
  vc_decl(max_corner)
  vc_decl(test_vec_1)
  vc_decl(test_vec_2)
  vc_decl(test_vec_3)
  vc_decl(yhat)

  declare_varp(polygon_x)
  declare_varp(polygon_temperature)
  declare_varp(polygon_recyc_coef)

```

```

declare_varp(polygon_points)
declare_varp(polygon_zone)
declare_varp(polygon_stratum)
declare_varp(polygon_material)
declare_varp(polygon_segment)

integer num_polygon, current_material, current_stratum, temp_material
integer face1, zone1, zone2, stratum, sector1, sector2, y_div, num_zone1, num_zone2
integer xcorner1, xcorner2, xcorner3, xcorner4, zcorner1, zcorner2, zcorner3, zcorner4
integer sect_zone1_1, sect_zone2_1
character*100 plot

vc_decl(x1)
vc_decl(x2)
<Memory allocation interface 0>
st_decls
vc_decls
gi_ext

<pardef.h 0>
<combal.h 0>
<comst.h 0>
<comgeo.h 0>
<compls.h 0>
<comflg.h 0>
<comrfl.h 0>
<comrat.h 0>
<comsv.h 0>
<comstat.h 0>
<compar.h 0>

datafile_diskin = diskin + 1
wallfile_diskin = diskin + 2

open(unit = diskin, file = filename, status = 'old', form = 'formatted')

/* read the file containing keywords that describe the way plotting has to be done */
/* Defaults and other initializations */
uedge = F
degas = F
sonnet = F
single_null = F
double_null = F

num_polygon = 0

minx = zero
minz = zero
mindensity = zero

var_realloca(polygon_x)
var_realloca(polygon_temperature)
var_realloca(polygon_recyc_coef)
var_realloca(polygon_points)
var_realloca(polygon_zone)
var_realloca(polygon_stratum)
var_realloca(polygon_material)

```

```

var_realloca(polygon_segment)

symmetry = geometry_symmetry_cylindrical
zone = 0
current_temperature = const(3., 2)
current_recyc_coef = one
wall_recyc = const(0.99)
pfr_recyc = const(0.99)
target_recyc = const(0.98)
facearray_0 = int_unused
facearray_1 = int_unused

loop1: continue
  assert(read_string(diskin, line, length))
  assert(length ≤ len(line))
  length = parse_string(line(: length))
  p = 0
  assert(next_token(line, b, e, p))
  keyword = line(b : e)
  assert(next_token(line, b, e, p))
  if(keyword ≡ 'symmetry') then
    if(line(b : e) ≡ 'cylindrical') then
      symmetry = geometry_symmetry_cylindrical
    else if(line(b : e) ≡ 'plane') then
      symmetry = geometry_symmetry_plane
    else if(line(b : e) ≡ 'oned') then
      symmetry = geometry_symmetry_oned
    else
      assert(line(b : e) ≡ 'cylindrical' ∨ line(b : e) ≡ 'plane')
    end if
    goto loop1
  end if
  assert(keyword ≡ 'degasfile' ∨ keyword ≡ 'uedgefile' ∨ keyword ≡ 'sonnetfile')
  datafilename = line(b : e)
  open(unit = datafile_diskin, file = datafilename, status = 'old', form = 'formatted')
  if(keyword ≡ 'degasfile') then
    degas = T
    call inpt(datafile_diskin, symmetry, xmin, xmax, zmin, zmax)
    call init_geometry
    close(unit = datafile_diskin)
  elseif(keyword ≡ 'uedgefile') then
    uedge = T
    nu = datafile_diskin
    vacuum = F
  elseif(keyword ≡ 'sonnetfile') then
    sonnet = T /* everything is the same as for uedge for single null */
    uedge = T
  endif
  if(¬uedge) then
    min_corner_1 = xmin
    min_corner_2 = zero
    min_corner_3 = zmin
    max_corner_1 = xmax

```

```

    if (symmetry ≡ geometry_symmetry_cylindrical) then
        max_corner2 = two * PI
    else
        max_corner2 = one
    end if
    max_corner3 = zmax
    call universal_cell(symmetry, min_corner, max_corner, vol)
end if

loop2: continue
if (¬read_string(diskin, line, length))
    goto eof
assert(length ≤ len(line))
length = parse_string(line(: length))
p = 0
assert(next_token(line, b, e, p))
keyword = line(b : e)
if (keyword ≡ 'null_type') then
    assert(next_token(line, b, e, p))
    null_type = read_integer(line(b : e))
    if (null_type ≡ 1) then
        single_null = T
    elseif (null_type ≡ 2) then
        double_null = T
        ix_mirror = 0 // initialise only
    else
        print *, 'unknown_null_type'
        assert(F)
    endif
    if (uedge ∧ ¬sonnet) then
        call inpt_uedge(nu, nx, ny, ixpt1, ixpt2, iysptrx, rm, zm, mx, my, null_type, ix_mirror)
    elseif (sonnet) then
        call inpt_sonnet(datafile_diskin, xmin, xmax, zmin, zmax, rm, zm, nx, ny, mx, my, ixpt1,
            ixpt2, iysptrx, null_type, ix_mirror)
    endif
    call init_geometry
elseif (keyword ≡ 'new_zone') then
    zone++
@#if 0
    zn_volume(zone) = zero
@#endif
assert(next_token(line, b, e, p))
new_zone_type = line(b : e)
elseif (keyword ≡ 'new_polygon') then
    n = 0
    increment_num_polygon
elseif (keyword ≡ 'define_polygon') then
    polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
    polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
    polygon_pointsnum_polygon = n
    polygon_zonenum_polygon = zone
    polygon_stratumnum_polygon = current_stratum
    polygon_materialnum_polygon = current_material

```

```

polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x_1, 0, num_polygon, zonearray, 1, facearray)
if (zn_volume(zone)  $\equiv$  real_undef) then
  zn_volume(zone) = zero
  zn_index(zone, zi_ix) = 0 // At least zi_ptr does get used now.
  zn_index(zone, zi_iz) = 0
  zn_index(zone, zi_iy) = 0
  zn_index(zone, zi_ptr) = zone
  if (new_zone_type  $\equiv$  'solid') then
    zn_type_set(zone, zn_solid)
  else if (new_zone_type  $\equiv$  'exit') then
    zn_type_set(zone, zn_exit)
  else if (new_zone_type  $\equiv$  'vacuum') then
    zn_type_set(zone, zn_vacuum)
  else if (new_zone_type  $\equiv$  'plasma') then
    zn_type_set(zone, zn_plasma)
  else if (new_zone_type  $\equiv$  'exit') then
    zn_type_set(zone, zn_exit)
  else
    write (stderr, *) 'Unknown_zone_type:', new_zone_type
    assert ( $\mathcal{F}$ )
  end if
  call set_zn_min_max(n, polygon_x_1, 0, num_polygon, zone,  $\mathcal{T}$ )
else
  call set_zn_min_max(n, polygon_x_1, 0, num_polygon, zone,  $\mathcal{F}$ )
end if
zn_volume(zone) = zn_volume(zone) + polygon_volume(n, polygon_x_1, 0, num_polygon)
elseif (keyword  $\equiv$  'wall') then
  ⟨ Wall Keyword 1.4 ⟩

elseif (keyword  $\equiv$  'outer') then
loop3: continue
if (next_token(line, b, e, p)) then
  i = read_integer(line(b:e))
  if (i  $\equiv$  0  $\vee$  i  $\equiv$  4) then
    polygon_x_1, n, num_polygon = xmin;
    polygon_x_2, n, num_polygon = zmin
  else if (i  $\equiv$  1) then
    polygon_x_1, n, num_polygon = xmin;
    polygon_x_2, n, num_polygon = zmax
  else if (i  $\equiv$  2) then
    polygon_x_1, n, num_polygon = xmax;
    polygon_x_2, n, num_polygon = zmax
  else if (i  $\equiv$  3) then
    polygon_x_1, n, num_polygon = xmax;
    polygon_x_2, n, num_polygon = zmin
  end if
  n++
  polygon_segment_n, num_polygon = 0 // i.e., not used
  assert (n  $\leq$  num_points - 1)

```

```

    goto loop3
  end if
elseif (keyword ≡ 'grid') then
  ⟨Grid Keyword 1.5⟩
elseif (keyword ≡ 'edge') then
  ⟨Edge Keyword 1.6⟩
elseif (keyword ≡ 'mindensity') then
  assert(next_token(line, b, e, p))
  mindensity = read_real(line(b:e))
elseif (keyword ≡ 'minz') then
  assert(next_token(line, b, e, p))
  minz = read_real(line(b:e))
elseif (keyword ≡ 'minx') then
  assert(next_token(line, b, e, p))
  minx = read_real(line(b:e))
elseif (keyword ≡ 'wallfile') then
  assert(next_token(line, b, e, p))
  wallfilename = line(b:e)
  open(unit = wallfile_diskin, file = wallfilename, status = 'old', form = 'formatted')
  read(wallfile_diskin, *) nowals
  read(wallfile_diskin, *) SP(nosegsxzi, i = 1, nowals)
  do i = 1, nowals
    do j = 1, nosegsxzi
      read(wallfile_diskin, *) xwallj, i, zwallj, i
    enddo
  enddo
  vacuum = T
elseif (keyword ≡ 'mesh') then
  ⟨Mesh Keyword 1.7⟩
elseif (keyword ≡ 'stratum') then
  assert(next_token(line, b, e, p))
  current_stratum = read_integer(line(b:e))
elseif (keyword ≡ 'material') then
  assert(next_token(line, b, e, p))
  current_material = ma_lookup(line(b:e))
elseif (keyword ≡ 'temperature') then
  assert(next_token(line, b, e, p))
  current_temperature = read_real(line(b:e))
elseif (keyword ≡ 'recyc_coef') then
  assert(next_token(line, b, e, p))
  current_recyc_coef = read_real(line(b:e))
  /* These three are presently used only for the UEDGE_RECT_SLAB geometry. */
elseif (keyword ≡ 'wall_recyc_coef') then
  assert(next_token(line, b, e, p))
  wall_recyc = read_real(line(b:e))
elseif (keyword ≡ 'target_recyc_coef') then
  assert(next_token(line, b, e, p))

```

```

    target_recyc = read_real(line(b : e))
elseif (keyword  $\equiv$  'pfr_recyc_coef') then
    assert(next_token(line, b, e, p))
    pfr_recyc = read_real(line(b : e))

    /* insert additional keywords below this */

elseif (keyword  $\equiv$  'end') then

    goto eof
else
    write(stderr, *) 'Unknown keyword', trim(keyword)
    assert( $\mathcal{F}$ )
end if
goto loop2
eof: continue
close (unit = diskin)

call boundaries_neighbors

/* Set up default sectors. These are taken to be at the interface between plasma / vacuum and solid
/ exit zones. The procedure consists of first cycling through all of the polygons of zone type solid
or exit and checking each surface to see if there is a plasma or vacuum zone on the other side. The
filter used here is that find_poly_zone arranges for the type of zone1 to match that of polygon i.
This is done so that, for example, a target sector is identified by having zone1 be a solid and
zone2 be plasma. The target sector is then associated with zone2 (and its corresponding face,
-face1); zone1 is passed to define_sector as a check. */
segment = 0 // UEDGE_RECT_SLAB
segment1 = 0 // Test of diagnostic sectors
y_div = 1 // Only axisymmetric problems here
do i = 1, num_polygon
  do j = 0, polygon_pointsi - 1
    if (polygon_x1, j, i  $\neq$  polygon_x1, j+1, i  $\vee$  polygon_x2, j, i  $\neq$  polygon_x2, j+1, i) then
      /* This used to be part of find_poly_zone. */
      vc_set(x1, polygon_x1, j, i, zero, polygon_x2, j, i)
      vc_set(x2, polygon_x1, j+1, i, zero, polygon_x2, j+1, i)
      face1 = lookup_surface(x1, x2)
      call find_poly_zone(face1, zn_type(polygon_zonei), polygon_zonei, y_div, sect_zone1,
        num_zone1, sect_zone2, num_zone2)
      if ((num_zone1 > 0)  $\wedge$  (num_zone2 > 0)) then
        assert(num_zone1  $\equiv$  1)
        assert(num_zone2  $\equiv$  1)
        zone1 = sect_zone11
        zone2 = sect_zone21
      @if (GEOMETRY  $\equiv$  UEDGE_RECT_SLAB)
        if (polygon_x1, j, i  $\leq$  zero  $\wedge$  polygon_x1, j+1, i  $\leq$  zero  $\wedge$  (polygon_x1, j+1, i >
          polygon_x1, j, i)  $\wedge$  polygon_x2, j, i  $\equiv$  const(0.75)  $\wedge$  polygon_x2, j+1, i  $\equiv$  const(0.75)) then
          assert(zn_type(zone1)  $\equiv$  zn_plasma  $\wedge$  zn_type(zone2)  $\equiv$  zn_plasma)
          stratum = 6
          segment++
          sector1 = define_sector(stratum, segment, face1, zone1, 0) // Null second zone
          temp_material = ma_lookup('mirror')
          define_sector_wall(sector1, temp_material, current_temperature * boltzmanns_const,
            current_recyc_coef)
          segment++

```

```

sector2 = define_sector(stratum, segment, -face1, zone2, 0)
define_sector_wall(sector2, temp_material, current_temperature * boltzmanns_const,
    current_recyc_coef)
/* Designate the rest of the surfaces along z = 0.75 as diagnostic sectors. The sector
numbers are stored in arrays for use in calls to diag_grp_init below. Note that in most
instances, defining sectors on one side of an interface is sufficient. A second group of
sectors with arguments -face1 and zone2 could have been defined, if needed. */
else if ((polygon_x1, j+1, i > polygon_x1, j, i) ^ polygon_x2, j, i ≡
    const(0.75) ^ polygon_x2, j+1, i ≡ const(0.75)) then
    assert(zn_type(zone1) ≡ zn_plasma ^ zn_type(zone2) ≡ zn_plasma)
    stratum = 7
    segment1 ++
    sector1 = define_sector(stratum, segment1, face1, zone1, 0)
    grp_sectors1[segment1] = sector1
    /* Verify that dimension of grp_sector array is adequate. */
    assert(segment1 ≤ mx)
end if
@#elif (GEOMETRY ≡ DEGAS_CMOD)
if ((zn_type(zone1) ≡ zn_vacuum ^ zn_type(zone2) ≡ zn_plasma) ∨ (zn_type(zone1) ≡
    zn_vacuum ^ zn_type(zone2) ≡ zn_vacuum ^ ¬(sc_check(int_lookup(-face1,
    sector_surface_1, nsectors)))) then
    stratum = 6
    segment1 ++
    sector1 = define_sector(stratum, segment1, -face1, zone2, zone1)
    grp_sectors1[segment1] = sector1
end if
@#endif
stratum = polygon_stratum_i
if ((zn_type(zone1) ≡ zn_solid) ^ (zn_type(zone2) ≡ zn_plasma)) then
    sector1 = define_sector(stratum, polygon_segment_j+1, i, -face1, zone2, zone1)
    define_sector_plasma(sector1)
    sector2 = define_sector(stratum, polygon_segment_j+1, i, face1, zone1, zone2)
    assert(ma_check(polygon_material_i))
    assert(polygon_temperature_i > zero)
    define_sector_target(sector2, polygon_material_i, polygon_temperature_i * boltzmanns_const,
        polygon_recyc_coef_i)

else if ((zn_type(zone1) ≡ zn_solid) ^ (zn_type(zone2) ≡ zn_vacuum)) then
    sector1 = define_sector(stratum, polygon_segment_j+1, i, -face1, zone2, zone1)
    define_sector_vacuum(sector1)
    sector2 = define_sector(stratum, polygon_segment_j+1, i, face1, zone1, zone2)
    assert(ma_check(polygon_material_i))
    assert(polygon_temperature_i > zero)
    define_sector_wall(sector2, polygon_material_i, polygon_temperature_i * boltzmanns_const,
        polygon_recyc_coef_i)

else if ((zn_type(zone1) ≡ zn_exit) ^ ((zn_type(zone2) ≡ zn_plasma))) then
    sector1 = define_sector(stratum, polygon_segment_j+1, i, -face1, zone2, zone1)
    define_sector_plasma(sector1)
    sector2 = define_sector(stratum, polygon_segment_j+1, i, face1, zone1, zone2)
    define_sector_exit(sector2)

else if ((zn_type(zone1) ≡ zn_exit) ^ ((zn_type(zone2) ≡ zn_vacuum))) then
    sector1 = define_sector(stratum, polygon_segment_j+1, i, -face1, zone2, zone1)

```

```

        define_sector_vacuum(sector1)
        sector2 = define_sector(stratum, polygon_segmentj+1, i, face1, zone1, zone2)
        define_sector_exit(sector2)
    else
        goto next_surface
    end if

end if
end if next_surface continue
end do
end do

call default_diag_setup

@#if (GEOMETRY ≡ UEDGE_RECT_SLAB ∨ GEOMETRY ≡ DEGAS_CMOD)
    /* Define this set of sectors as a separate “diagnostic group”. No energy or angle distribution is to
       be computed on these sectors: */
    var = sc_diag_unknown
    tab_index = 0
    var_min = zero
    var_max = zero
    mult = zero
    spacing = sc_diag_spacing_unknown
    call diag_grp_init('Throat_sectors', segment1, var, tab_index, var_min, var_max, mult, spacing,
        grp_sectors1)
@#endif

call end_sectors

call rg_detector_setup

call end_detectors

call check_geometry
call write_geometry
vola = zero
do i = 1, zn_num
    assert(zn_volume(i) > 0)
    vola = vola + zn_volume(i)
end do
write(stdout, *) 'vol_and_vola=', vol, vola

call erase_geometry

return end

```

Wall keyword section

```

⟨ Wall Keyword 1.4 ⟩ ≡
  assert(next_token(line, b, e, p))
  wall_no = read_integer(line(b : e))
  assert(next_token(line, b, e, p))
  complete = (line(b : e) ≡ '*')
  if (¬complete) then
    wall_start = read_integer(line(b : e))
    assert(next_token(line, b, e, p))
    wall_stop = read_integer(line(b : e))
  end if
  if (next_token(line, b, e, p)) then
    wall_direction = read_integer(line(b : e))
  else
    wall_direction = 1
  end if
  if (complete) then
    if (wall_direction > 0) then
      wall_start = 1
      wall_stop = nosegsxzwall_no + 1
    else
      wall_start = nosegsxzwall_no + 1
      wall_stop = 1
    end if
  end if
  if (wall_direction > 0) then
    if (wall_stop < wall_start)
      wall_stop = wall_stop + nosegsxzwall_no + 1
    else
      if (wall_stop > wall_start)
        wall_start = wall_start + nosegsxzwall_no + 1
      end if
    /* Use wall_no and kumat to set stratum and material numbers. This is intended as a
    default which can be superseded via the “stratum” or “material” keywords, if desired. Note that
    in general the material can vary from sector to sector. To allow that here would require a very fine
    polygon specification. Likewise, set temperature. Note that twall has a toroidal segment index,
    set to 1. There is an assertion on the temperature at the end. */
    current_stratum = wall_no
    current_material = ma_lookup(kumatwall_start, wall_no)
    if (twallwall_start, 1, wall_no > zero)
      current_temperature = twallwall_start, 1, wall_no // set current_recyc_coef from frabsorb?
  do i = wall_start, wall_stop, wall_direction
    polygon_x1, n, num_polygon = xwallmod(i-1, nosegsxzwall_no+1)+1, wall_no
    polygon_x2, n, num_polygon = xwallmod(i-1, nosegsxzwall_no+1)+1, wall_no
    n++
    polygon_segmentn, num_polygon = mod(i - 1, nosegsxzwall_no + 1) + 1
    assert(n ≤ num_points - 1) /* Check to see that other values of kumat for this polygon are
    consistent with the one set at wall_start. If not, set current_material to a special value which
    can either be trapped downstream, or superseded by use of the “current_material” keyword. */
  if (i ≠ nosegsxzwall_no + 1) then
    temp_material = ma_lookup(kumatwall_start, wall_no)
    if (current_material ≠ temp_material)
      current_material = -1

```

```
    end if  
  end do
```

This code is used in section 1.3.

Grid keyword section

⟨ Grid Keyword 1.5 ⟩ ≡

```

  assert(next_token(line, b, e, p))
  grid_startv = read_integer(line(b : e))
  assert(next_token(line, b, e, p))
  grid_stopv = read_integer(line(b : e))
  assert(next_token(line, b, e, p))
  grid_starth = read_integer(line(b : e))
  assert(next_token(line, b, e, p))
  grid_stoph = read_integer(line(b : e))
  if (next_token(line, b, e, p)) then
    grid_sense = read_integer(line(b : e))
  else
    grid_sense = 1
  end if
  do j2 = grid_starth, grid_stoph
    plot = '␣'
    do j1 = grid_startv, grid_stopv
      if (kzone1(j1, j2) ≤ 0 ∨ kzone2(j1, j2) ≤ 0) then
        plot(j1 : j1) = 'k'
        goto skip
      else if (denehvt(j1, j2, 1) ≤ mindensity) then
        plot(j1 : j1) = 'd'
        goto skip
      else if (gridx(j1, j2, 1) ≤ minx ∨ gridx(j1, j2 + 1, 1) ≤ minx ∨ gridx(j1 + 1, j2 + 1, 1) ≤ minx ∨ gridx(j1 + 1, j2, 1) ≤ minx) then
        plot(j1 : j1) = 'x'
        goto skip
      else if (gridz(j1, j2, 1) ≤ minz ∨ gridz(j1, j2 + 1, 1) ≤ minz ∨ gridz(j1 + 1, j2 + 1, 1) ≤ minz ∨ gridz(j1 + 1, j2, 1) ≤ minz) then
        plot(j1 : j1) = 'z'
        goto skip
      endif
    if (grid_sense ≡ 1) then
      x_test1, 0 = gridx(j1, j2, 1)
      x_test2, 0 = gridz(j1, j2, 1)
      x_test1, 1 = gridx(j1 + 1, j2, 1)
      x_test2, 1 = gridz(j1 + 1, j2, 1)
      x_test1, 2 = gridx(j1 + 1, j2 + 1, 1)
      x_test2, 2 = gridz(j1 + 1, j2 + 1, 1)
      x_test1, 3 = gridx(j1, j2 + 1, 1)
      x_test2, 3 = gridz(j1, j2 + 1, 1)
    else if (grid_sense ≡ 2) then
      x_test1, 0 = gridx(j1, j2, 1)
      x_test2, 0 = gridz(j1, j2, 1)
      x_test1, 1 = gridx(j1, j2 + 1, 1)
      x_test2, 1 = gridz(j1, j2 + 1, 1)
      x_test1, 2 = gridx(j1 + 1, j2 + 1, 1)
      x_test2, 2 = gridz(j1 + 1, j2 + 1, 1)
      x_test1, 3 = gridx(j1 + 1, j2, 1)
      x_test2, 3 = gridz(j1 + 1, j2, 1)
    else

```

```

    assert('Unexpected_value_of_grid_sense' == '□')
end if
x_test1,4 = x_test1,0
x_test2,4 = x_test2,0
vol_test = polygon_volume(4, x_test)
if (vol_test ≤ epsilon) then
    if (vol_test < -epsilon)
        write(stdout, *) 'Ignoring_negative_volume_zone_at_□j1,□j2=□', j1, j2
    goto skip
end if
/* write(stdout, '(5f10.4)') gridx(j1,j2,1),gridx(j1,j2+1,1), gridx(j1+1,j2+1,1),gridx(j1+
1,j2,1),gridx(j1,j2,1)write(stdout, '(5f10.4)')gridz(j1,j2,1),gridz(j1,j2+1,1),
gridz(j1+1,j2+1,1),gridz(j1+1,j2,1),gridz(j1,j2,1) */
zone++
increment_num_polygon
do n = 0, 4
    polygon_x1, n, num_polygon = x_test1, n
    polygon_x2, n, num_polygon = x_test2, n
end do
polygon_points_num_polygon = 4
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = current_stratum
polygon_material_num_polygon = current_material
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(4, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_plasma)
zn_volume(zone) = vol_test
zn_index(zone, 1) = j1
zn_index(zone, 2) = j2
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
/* This is only an approximation and assumes that the zone is convex */
vc_set(zone_center zone, const(0.25) * (polygon_x1, 0, num_polygon + polygon_x1, 1, num_polygon +
polygon_x1, 2, num_polygon + polygon_x1, 3, num_polygon), zero,
const(0.25) * (polygon_x2, 0, num_polygon + polygon_x2, 1, num_polygon + polygon_x2, 2, num_polygon +
polygon_x2, 3, num_polygon))
call set_zn_min_max(4, polygon_x1, 0, num_polygon, zone, T)
skip: continue
end do
write(stderr, *) j2, '□', plot(grid_startv : grid_stopv)
enddo

```

This code is used in section 1.3.

Edge keyword section

```

⟨Edge Keyword 1.6⟩ ≡
  assert(next_token(line, b, e, p))
  grid_startv = read_integer(line(b : e))
  assert(next_token(line, b, e, p))
  grid_stopv = read_integer(line(b : e))
  assert(next_token(line, b, e, p))
  grid_starth = read_integer(line(b : e))
  assert(next_token(line, b, e, p))
  grid_stoph = read_integer(line(b : e))
  assert(grid_startv ≡ grid_stopv ∨ grid_starth ≡ grid_stoph)
  if (grid_startv ≡ grid_stopv) then
    stepv = 0
  else if (grid_startv < grid_stopv) then
    stepv = 1
    num = grid_stopv - grid_startv + 1
  else
    stepv = -1
    num = grid_startv - grid_stopv + 1
  endif
  if (grid_starth ≡ grid_stoph) then
    steph = 0
  else if (grid_starth < grid_stoph) then
    steph = 1
    num = grid_stoph - grid_starth + 1
  else
    steph = -1
    num = grid_starth - grid_stoph + 1
  endif
  h = grid_starth
  v = grid_startv
  do i = 0, num - 1
    polygon_x1, n, num_polygon = gridx(v, h, 1)
    polygon_x2, n, num_polygon = gridz(v, h, 1)
    n++
    h = h + steph
    v = v + stepv
  enddo

```

This code is used in section 1.3.

Mesh keyword section

⟨ Mesh Keyword 1.7 ⟩ ≡

```

  assert(uedge)
  assert(next_token(line, b, e, p))
  xmin = read_real(line(b : e))
  assert(next_token(line, b, e, p))
  xmax = read_real(line(b : e))
  assert(next_token(line, b, e, p))
  zmin = read_real(line(b : e))
  assert(next_token(line, b, e, p))
  zmax = read_real(line(b : e))
  if (next_token(line, b, e, p)) then
    ymin = read_real(line(b : e))
    assert(next_token(line, b, e, p))
    ymax = read_real(line(b : e))
  else
    ymin = zero
    ymax = two * PI
  end if
  min_corner_1 = xmin
  min_corner_2 = ymin
  min_corner_3 = zmin
  max_corner_1 = xmax
  max_corner_2 = ymax
  max_corner_3 = zmax
@#if 0
  call init_geometry
@#endif
call universal_cell(symmetry, min_corner, max_corner, vol)
/* (xcorner1,zcorner1) is the left bottom corner of a cell. (2,2) (3,3) and (4,4) are the other corners
   if you proceed in the clock-wise direction around the computational rectangle. The convention
   varies, hence the corners have to be set to conform with the data file being used */
if (sonnet) then
  xcorner1 = 3;
  zcorner1 = 3
  xcorner2 = 1;
  zcorner2 = 1
  xcorner3 = 2;
  zcorner3 = 2
  xcorner4 = 4;
  zcorner4 = 4
else
  xcorner1 = 1;
  zcorner1 = 1
  xcorner2 = 3;
  zcorner2 = 3
  xcorner3 = 4;
  zcorner3 = 4
  xcorner4 = 2;
  zcorner4 = 2
endif /* A separate question is the direction around the physical rectangle. The polygon sent to
      decompose_polygon must be traversed in a clockwise direction in physical space. */

```

```

vc_set(yhat, zero, one, zero)
vc_set(test_vec_1, rm(1, 1, xcorner2) - rm(1, 1, xcorner1), zero, zm(1, 1, zcorner2) - zm(1, 1,
    zcorner1))
vc_set(test_vec_2, rm(1, 1, xcorner3) - rm(1, 1, xcorner2), zero, zm(1, 1, zcorner3) - zm(1, 1,
    zcorner2))
vc_cross(test_vec_1, test_vec_2, test_vec_3)
if (vc_product(test_vec_3, yhat) > zero) then
    grid_sense = 1
else if (vc_product(test_vec_3, yhat) < zero) then
    grid_sense = 2
else
    assert('First_cell_of_mesh_degenerate' ≡ ' ')
end if
do j1 = 1, nx
    do j2 = 1, ny
        zone++
        n = 0
        increment_num_polygon
        polygon_x_1, n, num_polygon = rm(j1, j2, xcorner1)
        polygon_x_2, n, num_polygon = zm(j1, j2, zcorner1);
        n++
        if (grid_sense ≡ 1) then
            polygon_x_1, n, num_polygon = rm(j1, j2, xcorner2)
            polygon_x_2, n, num_polygon = zm(j1, j2, zcorner2);
            n++
            polygon_x_1, n, num_polygon = rm(j1, j2, xcorner3)
            polygon_x_2, n, num_polygon = zm(j1, j2, zcorner3);
            n++
            polygon_x_1, n, num_polygon = rm(j1, j2, xcorner4)
            polygon_x_2, n, num_polygon = zm(j1, j2, zcorner4);
            n++
        else if (grid_sense ≡ 2) then
            polygon_x_1, n, num_polygon = rm(j1, j2, xcorner4)
            polygon_x_2, n, num_polygon = zm(j1, j2, zcorner4);
            n++
            polygon_x_1, n, num_polygon = rm(j1, j2, xcorner3)
            polygon_x_2, n, num_polygon = zm(j1, j2, zcorner3);
            n++
            polygon_x_1, n, num_polygon = rm(j1, j2, xcorner2)
            polygon_x_2, n, num_polygon = zm(j1, j2, zcorner2);
            n++
        else
            assert('grid_sense_improperly_set' ≡ ' ')
        end if
        polygon_x_1, n, num_polygon = polygon_x_1, 0, num_polygon
        polygon_x_2, n, num_polygon = polygon_x_2, 0, num_polygon
        polygon_points_num_polygon = n
        polygon_zone_num_polygon = zone
        polygon_stratum_num_polygon = current_stratum
        polygon_material_num_polygon = current_material
        polygon_temperature_num_polygon = current_temperature
        polygon_recyc_coef_num_polygon = current_recyc_coef
    
```

```

zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_plasma)
zn_volume(zone) = polygon_volume(n, polygon_x1, 0, num_polygon)
zn_index(zone, 1) = j1
zn_index(zone, 2) = j2
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
vc_set(zone_center zone, rm_j1, j2, 0, zero, zm_j1, j2, 0)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, T)
enddo
enddo

/* Two cases: vacuum and -vacuum; for the former, wallfile data must be specified. In both cases,
we use assumed values for the stratum numbers:

1. Inner target,
2. Outer target,
3. Wall along main SOL,
4. Core region (treated as an exit here),
5. Wall below private flux region.

The stratum numbers for plasma and vacuum regions are set to current_stratum; these numbers are
not used since the stratum for each sector is defined by the solid / exit stratum. The value of
current_material is used everywhere (and is, hence, constant). */
@#if (GEOMETRY ≠ UEDGE_RECT_SLAB)
if (-vacuum) then
    ⟨Not-Vacuum Boundary Section 1.8⟩
else
    ⟨Vacuum Boundary Section 1.9⟩
endif
    ⟨Core Boundary Section 1.10⟩
@#else
    ⟨UERS Boundary Section 1.11⟩
@#endif

/* END OF "MESH" KEYWORD SECTION */

```

This code is used in section 1.3.

Not-Vacuum Boundary Section. I.e., Mesh itself is the problem boundary.

⟨Not-Vacuum Boundary Section 1.8⟩ ≡

```

zone++ /* Start from left lower corner of box and go clockwise along box till right lower corner.
        Return along outer edge of SOL to starting pt. */
n = 0
increment_num_polygon // SINGLE NULL
if (single_null) then
    polygon_x1, n, num_polygon = xmin
    polygon_x2, n, num_polygon = zmin;
    n++
    polygon_x1, n, num_polygon = xmin
    polygon_x2, n, num_polygon = zmax;
    n++
    polygon_x1, n, num_polygon = xmax
    polygon_x2, n, num_polygon = zmax;
    n++
    polygon_x1, n, num_polygon = xmax
    polygon_x2, n, num_polygon = zmin;
    n++
    j1 = nx;
    j2 = ny
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner3)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner3);
    n++
    do j1 = nx, 1, -1
        polygon_segment_n, num_polygon = j1
        polygon_x1, n, num_polygon = rm(j1, j2, xcorner2)
        polygon_x2, n, num_polygon = zm(j1, j2, zcorner2);
        n++
    end do // DOUBLE NULL
elseif (double_null ∧ uedge) then // Why uedge ???
    polygon_x1, n, num_polygon = xmin
    polygon_x2, n, num_polygon = zmin;
    n++
    polygon_x1, n, num_polygon = xmin
    polygon_x2, n, num_polygon = zmax;
    n++
    polygon_x1, n, num_polygon = xmax
    polygon_x2, n, num_polygon = zmax;
    n++
    polygon_x1, n, num_polygon = xmax
    polygon_x2, n, num_polygon = zmin;
    n++
    j1 = nx;
    j2 = ny
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner3)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner3);
    n++
    j2 = ny
    do j1 = nx, ix_mirror + 1, -1
        polygon_segment_n, num_polygon = j1

```

```

    polygon_x1, n, num_polygon = rm(j1, j2, xcorner2)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner2);
    n++
enddo
j1 = ix_mirror + 1
do j2 = ny, 1, -1
    polygon_segment_n, num_polygon = j2
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
    n++
enddo
j1 = ix_mirror;
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
n++
j1 = ix_mirror
do j2 = 1, ny
    polygon_segment_n, num_polygon = j2
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner3)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner3);
    n++
enddo
j2 = ny
do j1 = ix_mirror, 1, -1
    polygon_segment_n, num_polygon = j1
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner2)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner2);
    n++
enddo
else
    print *, 'neither_single_nor_double_null'
    assert( $\mathcal{F}$ )
endif
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 3
polygon_material_num_polygon = current_material
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_solid)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, T)
/* Start from left lower corner of box and to left top of SOL to right top of SOL. Cut down from

```

there to bottom of box and then close the polygon. Note that in a strange configuration, this “cut” could slice back across previous polygon points, ruining its connectedness. */

```

n = 0
increment_num_polygon // SINGLE and DOUBLE NULL
polygon_x1, n, num_polygon = xmin
polygon_x2, n, num_polygon = zmin;
n++
j1 = 1
do j2 = ny, 1, -1
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner2)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner2);
  n++
  polygon_segment_n, num_polygon = j2
end do
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
n++
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 1
polygon_material_num_polygon = current_material
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_volume(zone) = zn_volume(zone) + polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, F)
/* Start again at bottom of box and proceed straight up to right top; proceed along PFR to left
bottom of SOL; go again straight down to bottom of box and then close the polygon. */
n = 0
increment_num_polygon
j2 = 1;
j1 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zmin;
n++
do j1 = 1, ixpt1
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
  n++
  polygon_segment_n, num_polygon = j1
end do
do j1 = ixpt2 + 1, nx
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
  n++

```

```

    polygon_segmentn, num_polygon = j1
end do
j1 = nx
polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
n++
polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_pointsnum_polygon = n
polygon_zonenum_polygon = zone
polygon_stratumnum_polygon = 5
polygon_materialnum_polygon = current_material
polygon_temperaturenum_polygon = current_temperature
polygon_recyc_coefnum_polygon = current_recyc_coef
zonearray0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_volume(zone) = zn_volume(zone) + polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone,  $\mathcal{F}$ ) /* Pick up at the bottom of the box
and go back up to the left bottom of the SOL, then go across to the right bottom of the SOL, to
the lower right bottom of the box and then close the polygon. */
n = 0
increment_num_polygon
j1 = nx;
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x2, n, num_polygon = zmin;
n++
do j2 = 1, ny
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
    n++
    polygon_segmentn, num_polygon = j2
end do
j2 = ny
polygon_x1, n, num_polygon = rm(j1, j2, xcorner3)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner3);
n++

polygon_x1, n, num_polygon = xmax
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_pointsnum_polygon = n
polygon_zonenum_polygon = zone
polygon_stratumnum_polygon = 2
polygon_materialnum_polygon = current_material
polygon_temperaturenum_polygon = current_temperature
polygon_recyc_coefnum_polygon = current_recyc_coef
zonearray0 = zone

```

```
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)  
zn_volume(zone) = zn_volume(zone) + polygon_volume(n, polygon_x1, 0, num_polygon)  
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone,  $\mathcal{F}$ )
```

This code is used in section 1.7.

Vacuum Boundary Section

⟨ Vacuum Boundary Section 1.9 ⟩ ≡

```

/* Specify a vacuum region // Start with top vacuum region */
if (single_null) then // SINGLE NULL
  zone++

  n = 0
  increment_num_polygon
  i = 2
  do j = nosegsxzi, 1, -1
    polygon-x1, n, num_polygon = xwallj, i
    polygon-x2, n, num_polygon = zwallj, i;
    n++
    polygon-segmentn, num_polygon = j
  end do
  j1 = nx;
  j2 = ny
  polygon-x1, n, num_polygon = rm(j1, j2, xcorner3)
  polygon-x2, n, num_polygon = zm(j1, j2, zcorner3);
  n++

  do j1 = nx, 1, -1
    polygon-segmentn, num_polygon = j1
    polygon-x1, n, num_polygon = rm(j1, j2, xcorner2)
    polygon-x2, n, num_polygon = zm(j1, j2, zcorner2);
    n++
  end do

  polygon-x1, n, num_polygon = polygon-x1, 0, num_polygon
  polygon-x2, n, num_polygon = polygon-x2, 0, num_polygon

  polygon-pointsnum_polygon = n
  polygon-zonenum_polygon = zone
  polygon-stratumnum_polygon = current-stratum
  polygon-materialnum_polygon = current-material
  polygon-temperaturenum_polygon = current-temperature
  polygon-recyc-coefnum_polygon = current-recyc-coef
  zonearray0 = zone
  call decompose_polygon(n, polygon-x1, 0, num_polygon, zonearray, 1, facearray)
  zn_type_set(zone, zn_vacuum)
  zn_index(zone, zi_ix) = 0
  zn_index(zone, zi_iz) = 0
  zn_index(zone, zi_iy) = 0
  zn_index(zone, zi_ptr) = zone
  zn_volume(zone) = polygon_volume(n, polygon-x1, 0, num_polygon)
  call set_zn_min_max(n, polygon-x1, 0, num_polygon, zone, T) /* Do top wall // Start from left
    lower corner of box and go clockwise along box till right lower corner. Return along top wall,
    touching the plasma corners (to complete the wall) at either end. */
  zone++

  n = 0
  increment_num_polygon
  polygon-x1, n, num_polygon = xmin
  polygon-x2, n, num_polygon = zmin;

```

```

n++
polygon-x1, n, num_polygon = xmin
polygon-x2, n, num_polygon = zmax;
n++
polygon-x1, n, num_polygon = xmax
polygon-x2, n, num_polygon = zmax;
n++
polygon-x1, n, num_polygon = xmax
polygon-x2, n, num_polygon = zmin;
n++

j1 = nx
j2 = ny
polygon-x1, n, num_polygon = rm(j1, j2, xcorner3)
polygon-x2, n, num_polygon = zm(j1, j2, zcorner3);
n++
polygon-segmentn, num_polygon = 0

i = 2
do j = 1, nosegsxzi
    polygon-x1, n, num_polygon = xwallj, i
    polygon-x2, n, num_polygon = zwallj, i;
    n++
    polygon-segmentn, num_polygon = j
end do

j1 = 1
j2 = ny
polygon-x1, n, num_polygon = rm(j1, j2, xcorner2)
polygon-x2, n, num_polygon = zm(j1, j2, zcorner2);
n++

polygon-x1, n, num_polygon = polygon-x1, 0, num_polygon
polygon-x2, n, num_polygon = polygon-x2, 0, num_polygon

polygon-pointsnum_polygon = n
polygon-zonenum_polygon = zone
polygon-stratumnum_polygon = 3
polygon-materialnum_polygon = current_material
polygon-temperaturenum_polygon = current_temperature
polygon-recyc_coefnum_polygon = current_recyc_coef
zonearray0 = zone
call decompose_polygon(n, polygon-x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_solid)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon-x1, 0, num_polygon)
call set_zn_min_max(n, polygon-x1, 0, num_polygon, zone, T) /* Outer vacuum region. // Start at
top (end) of outer wall segment (wall 2), go down to its beginning point, jump to right bottom
of SOL. Go up along right side of mesh to mirror point and finish at end of wall 2. */
else if (double_null ∧ uedge) then // DOUBLE NULL

zone++

```

```

n = 0
increment_num_polygon
i = 2
do j = nosegsxzi, 1, -1
  polygon_x1, n, num_polygon = xwallj, i
  polygon_x2, n, num_polygon = zwallj, i;
  n++
  polygon_segmentn, num_polygon = j
end do
j1 = nx;
j2 = ny
polygon_x1, n, num_polygon = rm(j1, j2, xcorner3)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner3);
n++

do j1 = nx, ix_mirror + 1, -1
  polygon_segmentn, num_polygon = j1
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner2)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner2);
  n++
end do

polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon

polygon_pointsnum_polygon = n
polygon_zonenum_polygon = zone
polygon_stratumnum_polygon = current_stratum
polygon_materialnum_polygon = current_material
polygon_temperaturenum_polygon = current_temperature
polygon_recyc_coefnum_polygon = current_recyc_coef
zonearray0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_vacuum)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, T)
/* Inner vacuum region. // Start at bottom (end) of inner wall segment, go up to its beginning
point, jump to inner mirror point of mesh, follow the mesh down to the left top. */
zone++

n = 0
increment_num_polygon
i = 3
do j = nosegsxzi, 1, -1
  polygon_x1, n, num_polygon = xwallj, i
  polygon_x2, n, num_polygon = zwallj, i;
  n++
  polygon_segmentn, num_polygon = j
end do
j1 = ix_mirror;
j2 = ny

```

```

polygon-x1, n, num_polygon = rm(j1, j2, xcorner3)
polygon-x2, n, num_polygon = zm(j1, j2, zcorner3);
n++

do j1 = ix_mirror, 1, -1
  polygon-segmentn, num_polygon = j1
  polygon-x1, n, num_polygon = rm(j1, j2, xcorner2)
  polygon-x2, n, num_polygon = zm(j1, j2, zcorner2);
  n++
end do

polygon-x1, n, num_polygon = polygon-x1, 0, num_polygon
polygon-x2, n, num_polygon = polygon-x2, 0, num_polygon

polygon-pointsnum_polygon = n
polygon-zonenum_polygon = zone
polygon-stratumnum_polygon = current_stratum
polygon-materialnum_polygon = current_material
polygon-temperaturenum_polygon = current_temperature
polygon-recyc-coefnum_polygon = current_recyc-coef
zonearray0 = zone
call decompose_polygon(n, polygon-x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_vacuum)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon-x1, 0, num_polygon)
call set_zn_min_max(n, polygon-x1, 0, num_polygon, zone, T)

/* Outer solid. // Start at top right of box, go to bottom right. Then, jump to right bottom of
SOL. Connect to wall 2 and follow it from beginning to end. */

zone++

n = 0
increment_num_polygon
polygon-x1, n, num_polygon = xmax
polygon-x2, n, num_polygon = zmax;
n++
polygon-x1, n, num_polygon = xmax
polygon-x2, n, num_polygon = zmin;
n++

j1 = nx
j2 = ny
polygon-x1, n, num_polygon = rm(j1, j2, xcorner3)
polygon-x2, n, num_polygon = zm(j1, j2, zcorner3);
n++
polygon-segmentn, num_polygon = 0

i = 2
do j = 1, nosegsxzi
  polygon-x1, n, num_polygon = xwallj, i
  polygon-x2, n, num_polygon = zwallj, i;
  n++
  polygon-segmentn, num_polygon = j

```

end do

```

polygon-x1, n, num_polygon = polygon-x1, 0, num_polygon
polygon-x2, n, num_polygon = polygon-x2, 0, num_polygon

```

```

polygon_pointsnum_polygon = n
polygon_zonenum_polygon = zone
polygon_stratumnum_polygon = 3
polygon_materialnum_polygon = current_material
polygon_temperaturenum_polygon = current_temperature
polygon_recyc_coefnum_polygon = current_recyc_coef
zonearray0 = zone

```

```

call decompose_polygon(n, polygon-x1, 0, num_polygon, zonearray, 1, facearray)

```

```

zn_type_set(zone, zn_solid)

```

```

zn_index(zone, zi_ix) = 0

```

```

zn_index(zone, zi_iz) = 0

```

```

zn_index(zone, zi_iy) = 0

```

```

zn_index(zone, zi_ptr) = zone

```

```

zn_volume(zone) = polygon_volume(n, polygon-x1, 0, num_polygon)

```

```

call set_zn_min_max(n, polygon-x1, 0, num_polygon, zone, T)

```

```

/* Inner solid. // Start at bottom left of box. Go to top left. Jump to start of wall 3 and follow
along its length. Step to left top of SOL and then finish polygon. */

```

```

zone++

```

```

n = 0

```

```

increment_num_polygon

```

```

polygon-x1, n, num_polygon = xmin

```

```

polygon-x2, n, num_polygon = zmin;

```

```

n++

```

```

polygon-x1, n, num_polygon = xmin

```

```

polygon-x2, n, num_polygon = zmax;

```

```

n++

```

```

i = 3

```

```

do j = 1, nosegsxzi

```

```

    polygon-x1, n, num_polygon = xwallj, i

```

```

    polygon-x2, n, num_polygon = zwallj, i;

```

```

    n++

```

```

    polygon_segmentn, num_polygon = j

```

```

end do

```

```

j1 = 1

```

```

j2 = ny

```

```

polygon-x1, n, num_polygon = rm(j1, j2, xcorner2)

```

```

polygon-x2, n, num_polygon = zm(j1, j2, zcorner2);

```

```

n++

```

```

polygon-x1, n, num_polygon = polygon-x1, 0, num_polygon

```

```

polygon-x2, n, num_polygon = polygon-x2, 0, num_polygon

```

```

polygon_pointsnum_polygon = n

```

```

polygon_zonenum_polygon = zone

```

```

polygon_stratumnum_polygon = 3

```

```

polygon_materialnum_polygon = current_material

```

```

polygon_temperaturenum_polygon = current_temperature

```

```

polygon_recyc_coefnum_polygon = current_recyc_coef
zonearray0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_solid)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, T)

/* Mirror boundary. // Start at top left of box and go to the top right. Pick up the last point of
wall 2 and proceed to the outer edge of the mirror cut on the mesh. Work along the mirror cut
(going across the core, too). Pick up the first point of wall 3 and finish the polygon. To avoid
having decompose_polygon trying to work with very oddly shaped polygons with segments from
the inner and outer leg, split this up into three pieces with connections to the universal cell, just
as is done below for the bottom wall. There shouldn't be nearly as much problem with it here.
Right piece */

zone++

n = 0
increment_num_polygon

polygon_x1, n, num_polygon = xmax
polygon_x2, n, num_polygon = zmax;
n++

i = 2
j = nosegszi
polygon_x1, n, num_polygon = xwallj, i
polygon_x2, n, num_polygon = zwallj, i;
n++

j2 = ny
j1 = ix_mirror + 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner2)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner2);
n++

j1 = ix_mirror + 1
do j2 = ny, 1, -1
    polygon_segmentn, num_polygon = j2
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
    n++
enddo
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zmax;
n++

polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon

polygon_pointsnum_polygon = n
polygon_zonenum_polygon = zone

```

```

polygon_stratum_num_polygon = 6
polygon_material_num_polygon = ma.lookup('mirror')
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x_1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_solid)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x_1, 0, num_polygon)
call set_zn_min_max(n, polygon_x_1, 0, num_polygon, zone, T)

/* Middle piece */
n = 0
increment_num_polygon

j1 = ix_mirror + 1;
j2 = 1
polygon_x_1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x_2, n, num_polygon = zmax;
n++
polygon_x_1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x_2, n, num_polygon = zm(j1, j2, zcorner1);
n++

j1 = ix_mirror;
j2 = 1
polygon_x_1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x_2, n, num_polygon = zm(j1, j2, zcorner4);
n++

polygon_x_1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x_2, n, num_polygon = zmax;
n++

polygon_x_1, n, num_polygon = polygon_x_1, 0, num_polygon
polygon_x_2, n, num_polygon = polygon_x_2, 0, num_polygon

polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 6
polygon_material_num_polygon = ma.lookup('mirror')
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x_1, 0, num_polygon, zonearray, 1, facearray)
zn_volume(zone) = zn_volume(zone) + polygon_volume(n, polygon_x_1, 0, num_polygon)
call set_zn_min_max(n, polygon_x_1, 0, num_polygon, zone, F)

/* Left piece */
n = 0
increment_num_polygon

j1 = ix_mirror;
j2 = 1

```

```

    polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
    polygon_x2, n, num_polygon = zmax;
    n++

    polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
    n++
    j1 = ix_mirror
    do j2 = 1, ny
        polygon_segment_n, num_polygon = j2
        polygon_x1, n, num_polygon = rm(j1, j2, xcorner3)
        polygon_x2, n, num_polygon = zm(j1, j2, zcorner3);
        n++
    enddo

    i = 3
    j = 1
    polygon_x1, n, num_polygon = xwall_j, i
    polygon_x2, n, num_polygon = zwall_j, i;
    n++

    polygon_x1, n, num_polygon = xmin;
    polygon_x2, n, num_polygon = zmax;
    n++

    polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
    polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon

    polygon_points_num_polygon = n
    polygon_zone_num_polygon = zone
    polygon_stratum_num_polygon = 6
    polygon_material_num_polygon = ma.lookup('mirror')
    polygon_temperature_num_polygon = current_temperature
    polygon_recyc_coef_num_polygon = current_recyc_coef
    zonearray_0 = zone
    call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
    zn_volume(zone) = zn_volume(zone) + polygon_volume(n, polygon_x1, 0, num_polygon)
    call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, F)

else
    assert('neither_single_nor_double_null' ≡ ' ')
end if /* Bottom wall // Start from left lower corner of box, go to left top of SOL, then to right
top of SOL. At this point, we cut straight down to the bottom of the box. Note that this arbitrary
cut between the strata makes some assumptions about the mesh shapes; a bizarre geometry could
end up with this polygon cutting across itself. */
zone++

n = 0
increment_num_polygon // SINGLE and DOUBLE NULL
polygon_x1, n, num_polygon = xmin
polygon_x2, n, num_polygon = zmin;
n++

j1 = 1
do j2 = ny, 1, -1
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner2)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner2);

```

```

    n++
    polygon_segmentn, num_polygon = j2
end do
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
n++
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_pointsnum_polygon = n
polygon_zonenum_polygon = zone
polygon_stratumnum_polygon = 1
polygon_materialnum_polygon = current_material
polygon_temperaturenum_polygon = current_temperature
polygon_recyc_coefnum_polygon = current_recyc_coef
zonearray0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_solid)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, T) /* Go straight up again to the right
top SOL and proceed to the first wall; follow it and then touch the left bottom of the SOL. Here
again we go straight down to the bottom of the box before completing the polygon. */
n = 0
increment_num_polygon
j1 = 1;
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
n++
polygon_segmentn, num_polygon = 0
i = 1
do j = 1, nosegszi
    polygon_x1, n, num_polygon = xwallj, i
    polygon_x2, n, num_polygon = zwallj, i;
    n++
    polygon_segmentn, num_polygon = j
end do
j1 = nx;
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
n++

```

```

polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 5
polygon_material_num_polygon = current_material
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_volume(zone) = zn_volume(zone) + polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone,  $\mathcal{F}$ )
/* Pick up again at the bottom and return to the left bottom of the SOL. Follow along the bottom
SOL; go to the bottom of the box and then close the polygon. */
n = 0
increment_num_polygon
j1 = nx;
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x2, n, num_polygon = zmin;
n++
j1 = nx
do j2 = 1, ny
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
  n++
  polygon_segment_n, num_polygon = j2
end do
j2 = ny
polygon_x1, n, num_polygon = rm(j1, j2, xcorner3)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner3);
n++

polygon_x1, n, num_polygon = xmax
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 2
polygon_material_num_polygon = current_material
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_volume(zone) = zn_volume(zone) + polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone,  $\mathcal{F}$ ) /* Private vacuum region */
zone++
n = 0

```

```

increment_num_polygon
j2 = 1
do j1 = 1, ixpt1
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
  n++
  polygon_segment_n, num_polygon = j1
end do
do j1 = ixpt2 + 1, nx
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
  n++
  polygon_segment_n, num_polygon = j1
end do
j1 = nx
polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
n++
i = 1
do j = nosegsxz_i, 1, -1
  polygon_segment_n, num_polygon = j
  polygon_x1, n, num_polygon = xwall_j, i
  polygon_x2, n, num_polygon = zwall_j, i;
  n++
end do
polygon_segment_n, num_polygon = 0
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = current_stratum
polygon_material_num_polygon = current_material
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_vacuum)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, T)

```

This code is used in section 1.7.

Core Section

⟨ Core Boundary Section 1.10 ⟩ ≡

```

zone++ /* Core region */
n = 0
increment_num_polygon
j2 = 1
if (single_null) then
  do j1 = ixpt1 + 1, ixpt2
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
    n++
    polygon_segment_n, num_polygon = j1
  enddo
  polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
  polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
  polygon_points_num_polygon = n
  polygon_zone_num_polygon = zone
  polygon_stratum_num_polygon = 4
  polygon_material_num_polygon = current_material
  polygon_temperature_num_polygon = current_temperature
  polygon_recyc_coef_num_polygon = current_recyc_coef
  zonearray_0 = zone
  call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
  zn_type_set(zone, zn_exit)
  zn_index(zone, zi_ix) = 0
  zn_index(zone, zi_iz) = 0
  zn_index(zone, zi_iy) = 0
  zn_index(zone, zi_ptr) = zone
elseif (double_null ∧ uedge) then
  j1 = ix_mirror;
  j2 = 1
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
  n++
  j1 = ix_mirror + 1;
  j2 = 1
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
  n++
  j2 = 1
  do j1 = ix_mirror + 1, ixpt2
    polygon_segment_n, num_polygon = j1
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
    n++
  enddo
  j2 = 1
  do j1 = ixpt1 + 1, ix_mirror
    polygon_segment_n, num_polygon = j1
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);

```

```

    n++
enddo

    polygon-x1, n, num_polygon = polygon-x1, 0, num_polygon
    polygon-x2, n, num_polygon = polygon-x2, 0, num_polygon
    polygon-pointsnum_polygon = n
    polygon-zonenum_polygon = zone
    polygon-stratumnum_polygon = 4
    polygon-materialnum_polygon = current_material
    polygon-temperaturenum_polygon = current_temperature
    polygon-recyc-coefnum_polygon = current-recyc-coef
    zonearray0 = zone
    call decompose_polygon(n, polygon-x1, 0, num_polygon, zonearray, 1, facearray)
    zn_type_set(zone, zn_exit)
    zn_index(zone, zi_ix) = 0
    zn_index(zone, zi_iz) = 0
    zn_index(zone, zi_iy) = 0
    zn_index(zone, zi_ptr) = zone
endif

    zn_volume(zone) = polygon_volume(n, polygon-x1, 0, num_polygon)
    call set_zn_min_max(n, polygon-x1, 0, num_polygon, zone, T)

```

This code is used in section 1.7.

UEDGE Rectangular Slab Boundary Zones

⟨ UERS Boundary Section 1.11 ⟩ ≡

```

/* These are the boundaries of Rensink's slab-ized single-null geometry (the outer half anyway).
   Each polygon can be associated with a portion of a real single-null geometry. This first piece would
   be the symmetry plane. */
zone++
n = 0
increment_num_polygon
polygon_x1, n, num_polygon = xmin
polygon_x2, n, num_polygon = zmin;
n++
j1 = 1;
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
n++
do j2 = 1, ny
  polygon_segment_n, num_polygon = j2
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner2)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner2);
  n++
end do
polygon_x1, n, num_polygon = xmax
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 1
polygon_material_num_polygon = ma_lookup('mirror')
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_solid)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, T)

/* Right piece - corresponds to the outer wall. */
n = 0
increment_num_polygon
polygon_x1, n, num_polygon = xmax
polygon_x2, n, num_polygon = zmin;
n++
j1 = 1;
j2 = ny
polygon_x1, n, num_polygon = rm(j1, j2, xcorner2)

```

```

polygon_x2, n, num_polygon = zm(j1, j2, zcorner2);
n++
do j1 = 1, nx
  polygon_segment_n, num_polygon = j1
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner3)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner3);
  n++
end do
polygon_x1, n, num_polygon = xmax
polygon_x2, n, num_polygon = zmax;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 2
polygon_material_num_polygon = ma_lookup('mo')
polygon_temperature_num_polygon = const(2.5, -2) /* eV, from EIRENE */
*electron_charge / boltzmanns_const // convert to K
polygon_recyc_coef_num_polygon = wall_recyc
zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_volume(zone) += polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone,  $\mathcal{F}$ )

/* Lower left piece - corresponds to core plasma (hence, is an exit). Note that this and the next
   piece are triangles overall instead of rectangles. This was done because of a roundoff error problem
   in matching surfaces with the universal cell. That task is now successfully accomplished with an
   additional (physically irrelevant piece). */
zone++

n = 0
increment_num_polygon
j1 = ixpt2;
j2 = 1
do j1 = ixpt2, 1, -1
  polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
  polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
  n++
  polygon_segment_n, num_polygon = j1
end do
j1 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
n++
polygon_x1, n, num_polygon = xmin
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 3

```

```

polygon_material_num_polygon = ma_lookup('Ex')
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x_1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_exit)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x_1, 0, num_polygon)
call set_zn_min_max(n, polygon_x_1, 0, num_polygon, zone, T)

/* Upper left piece - corresponds to the private flux boundary. */
zone++

n = 0
increment_num_polygon
polygon_x_1, n, num_polygon = xmin
polygon_x_2, n, num_polygon = zmax;
n++
j1 = nx;
j2 = 1
polygon_x_1, n, num_polygon = rm(j1, j2, xcorner4)
polygon_x_2, n, num_polygon = zm(j1, j2, zcorner4);
n++
do j1 = nx, ixpt2 + 1, -1
    polygon_segment_n, num_polygon = j1
    polygon_x_1, n, num_polygon = rm(j1, j2, xcorner1)
    polygon_x_2, n, num_polygon = zm(j1, j2, zcorner1);
    n++
end do
polygon_x_1, n, num_polygon = polygon_x_1, 0, num_polygon
polygon_x_2, n, num_polygon = polygon_x_2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 4
polygon_material_num_polygon = ma_lookup('mo')
polygon_temperature_num_polygon = const(2.5, -2) /* eV, from EIRENE */
*electron_charge / boltzmanns_const // convert to K
polygon_recyc_coef_num_polygon = pfr_recyc
zonearray_0 = zone
call decompose_polygon(n, polygon_x_1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_solid)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x_1, 0, num_polygon)
call set_zn_min_max(n, polygon_x_1, 0, num_polygon, zone, T)

/* This polygon fills in between the previous two and the universal cell. This is needed just to make
the geometry complete. */
zone++

```

```

n = 0
increment_num_polygon
polygon_x1, n, num_polygon = xmin
polygon_x2, n, num_polygon = zmax;
n++
j1 = ixpt2 + 1;
j2 = 1
polygon_x1, n, num_polygon = rm(j1, j2, xcorner1)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner1);
n++
polygon_x1, n, num_polygon = xmin
polygon_x2, n, num_polygon = zmin;
n++
polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points_num_polygon = n
polygon_zone_num_polygon = zone
polygon_stratum_num_polygon = 4
polygon_material_num_polygon = ma_lookup('mirror')
polygon_temperature_num_polygon = current_temperature
polygon_recyc_coef_num_polygon = current_recyc_coef
zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_solid)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, T)

/* Top piece - represents target plate. */
zone++

n = 0
increment_num_polygon
polygon_x1, n, num_polygon = xmax
polygon_x2, n, num_polygon = zmax;
n++
j1 = nx;
j2 = ny
polygon_x1, n, num_polygon = rm(j1, j2, xcorner3)
polygon_x2, n, num_polygon = zm(j1, j2, zcorner3);
n++
do j2 = ny, 1, -1
    polygon_segment_n, num_polygon = j2
    polygon_x1, n, num_polygon = rm(j1, j2, xcorner4)
    polygon_x2, n, num_polygon = zm(j1, j2, zcorner4);
    n++
end do
polygon_x1, n, num_polygon = xmin
polygon_x2, n, num_polygon = zmax;
n++

```

```

polygon_x1, n, num_polygon = polygon_x1, 0, num_polygon
polygon_x2, n, num_polygon = polygon_x2, 0, num_polygon
polygon_points num_polygon = n
polygon_zone num_polygon = zone
polygon_stratum num_polygon = 5
polygon_material num_polygon = ma_lookup('mo')
polygon_temperature num_polygon = const(1., -1) /* eV, from EIRENE */
*electron_charge / boltzmanns_const // convert to K
polygon_recyc_coef num_polygon = target_recyc
zonearray_0 = zone
call decompose_polygon(n, polygon_x1, 0, num_polygon, zonearray, 1, facearray)
zn_type_set(zone, zn_solid)
zn_index(zone, zi_ix) = 0
zn_index(zone, zi_iz) = 0
zn_index(zone, zi_iy) = 0
zn_index(zone, zi_ptr) = zone
zn_volume(zone) = polygon_volume(n, polygon_x1, 0, num_polygon)
call set_zn_min_max(n, polygon_x1, 0, num_polygon, zone, T)

```

This code is used in section 1.7.

Geometry-specific code specifying detector views. Actual code sections follow immediately below (FWEB insisted things be arranged this way).

⟨Functions and Subroutines 1.2⟩ +≡

@#if (*GEOMETRY* ≡ *BOX*)

⟨Box Detector Setup 1.13⟩

@#elif (*GEOMETRY* ≡ *WISING_CMOD* ∨ *GEOMETRY* ≡ *DEGAS_CMOD*)

⟨Wising C-Mod Detector Setup 1.15⟩

@#else

⟨Null Detector Setup 1.17⟩

@#endif

Detector views for 1-by-1 box (e.g., from *boxgen*).

```
"readgeometry.f" 1.13 ≡
  @m box_views 5 // To be used as a local dimension

⟨ Box Detector Setup 1.13 ⟩ ≡
  subroutine rg_detector_setup

    implicit_none_f77
    de_common
    zn_common
    implicit_none_f90

    ⟨ Memory allocation interface 0 ⟩

    detector_total_views = box_views
    var_alloc(de_view_points)
    var_alloc(de_view_algorithm)
    var_alloc(de_view_halfwidth)
    var_alloc(de_zone_frags)

    de_grps = 0
    de_view_size = 0
    var_alloc(de_view_tab)

    call rg_detector_setup_a

    return
  end
```

See also section 1.14.

This code is used in section 1.12.

Extension of the above subroutine. Statements actually making assignments to the detector pointer arrays (*de_zone_frgs* specifically) need to be separated from their allocation above so that their array indexing gets handled correctly.

⟨ Box Detector Setup 1.13 ⟩ +≡

```

subroutine rg_detector_setup_a
    define_varp(zone_frgs, FLOAT, de_zone_ind)

    implicit_none_f77
    de_common
    zn_common
    implicit_none_f90

    integer view, num, var, tab_index, spacing, i, zone
    integer grp_views box_views
    real var_min, var_max, mult

    declare_varp(zone_frgs)

    ⟨ Memory allocation interface 0 ⟩

    var_alloc(zone_frgs)

    if (detector_total_views > 0) then
        do view = 1, detector_total_views
            de_view_algorithmview = de_algorithm_uniform
            de_view_halfwidthview = const(3.) * PI / const(1.8, 2) // arbitrary
            vc_set(de_view_pointsview,de_view_start, one, zero, half)
        end do

        vc_set(de_view_points1,de_view_end, half, zero, zero)
        vc_set(de_view_points2,de_view_end, zero, zero, const(0.25))
        vc_set(de_view_points3,de_view_end, zero, zero, half)
        vc_set(de_view_points4,de_view_end, zero, zero, const(0.75))
        vc_set(de_view_points5,de_view_end, half, zero, one)

        do view = 1, detector_total_views
            call detector_view_setup(vc_args(de_view_pointsview,de_view_start), de_view_halfwidthview,
                de_view_algorithmview, zone_frgs)
            do zone = 1, zn_num
                de_zone_frgszone,view = zone_frgszone
            end do
        end do

        var_free(zone_frgs)

        num = detector_total_views
        var = de_var_unknown
        tab_index = zero
        var_min = zero
        var_max = zero
        mult = zero
        spacing = de_spacing_unknown
        do i = 1, num
            grp_viewsi = i
        end do
        call de_grp_init('Chord□integrals', num, var, tab_index, var_min, var_max, mult, spacing,
            grp_views)

```

```

    num = 1
    var = de_var_wavelength
    tab_index = 200
    var_min = const(6558.)
    var_max = const(6564.)
    mult = const(1., -10)
    spacing = de_spacing_linear
    grp_views_1 = 3
    call de_grp_init('Halpha_spectrum', num, var, tab_index, var_min, var_max, mult, spacing,
                   grp_views)
end if

return
end

```

Detector views for Wising's UEDGE simulation of C-Mod.

```

"readgeometry.f" 1.15 ≡
  @m ctop_views 35 // To be used as a local dimension.

```

```

⟨Wising C-Mod Detector Setup 1.15⟩ ≡
  subroutine rg_detector_setup
    implicit_none_f77
    de_common
    zn_common
    implicit_none_f90

    ⟨Memory allocation interface 0⟩

    detector_total_views = ctop_views + 1 // Spector chord is different
    var_alloc(de_view_points)
    var_alloc(de_view_algorithm)
    var_alloc(de_view_halfwidth)
    var_alloc(de_zone_fragments)

    de_grps = 0
    de_view_size = 0
    var_alloc(de_view_tab)

    call rg_detector_setup_a

    return
  end

```

See also section 1.16.

This code is used in section 1.12.

Extension of the above subroutine. Statements actually making assignments to the detector pointer arrays (*de_zone_frgs* specifically) need to be separated from their allocation above so that their array indexing gets handled correctly.

⟨Wising C-Mod Detector Setup 1.15⟩ +≡

subroutine *rg_detector_setup_a*

define_varp(zone_frgs, FLOAT, de_zone_ind)

implicit_none_f77

de_common

zn_common

implicit_none_f90

integer *view, num, var, tab_index, spacing, i, zone*

integer *grp_views ctop_views*

real *var_min, var_max, mult, r0ha, z0ha, r_end, z_end*

real *theta ctop_views*

data (*theta(i), i = 1, ctop_views*) /*dconst*(-1.59500, 1), *dconst*(-1.50800, 1), *dconst*(-1.41000, 1), *dconst*(-1.32200, 1), *dconst*(-1.22200, 1), *dconst*(-1.12700, 1), *dconst*(-1.03200, 1), *dconst*(-9.35501), *dconst*(-8.39001), *dconst*(-7.46500), *dconst*(-6.54001), *dconst*(-5.51001), *dconst*(-4.48001), *dconst*(-3.48999), *dconst*(-2.50000), *dconst*(-1.66699), *dconst*(-8.34015, -1), *dconst*(2.63000, -1), *dconst*(1.35999), *dconst*(2.50500), *dconst*(3.64999), *dconst*(4.53000), *dconst*(5.41000), *dconst*(6.33499), *dconst*(7.26001), *dconst*(8.28500), *dconst*(9.31000), *dconst*(1.02650, 1), *dconst*(1.12200, 1), *dconst*(1.21700, 1), *dconst*(1.31200, 1), *dconst*(1.41000, 1), *dconst*(1.50800, 1), *dconst*(1.59500, 1), *dconst*(1.69100, 1)/

declare_varp(zone_frgs)

⟨Memory allocation interface 0⟩

var_alloc(zone_frgs)

/ This is the vertex of the C-top array. In the coordinates used by the experimentalists, R₀ = 0.7517, Z₀ = 0.477; in these coordinates, the inner limiter is at R_{lim} = 0.44 and the top of the vacuum vessel is at Z_{max} = 0.60. Here, R_{lim} = 0.43, Z_{max} = 1.30115. */*

@#if (*GEOMETRY ≡ WISING_CM0D*)

r0ha = const(7.417, -1) // Vertex, from C-Mod guys

z0ha = const(1.17815)

@#elif (*GEOMETRY ≡ DEGAS_CM0D*)

r0ha = const(7.517, -1) // Shifted for old DEGAS R_{lim} = 0.44,

z0ha = const(1.087) // Z_{max} = 1.21.

@#endif

if (*detector_total_views > 0*) **then**

do *view = 1, ctop_views*

de_view_algorithm_{view} = de_algorithm_uniform

*de_view_halfwidth_{view} = half * PI / const(1.8, 2) // roughly distance between chords*

vc_set(de_view_points_{view, de_view_start}, r0ha, zero, z0ha)

z_end = zero

*r_end = r0ha + (z0ha - z_end) * tan(theta_{view} * PI / const(1.8, 2))*

vc_set(de_view_points_{view, de_view_end}, r_end, zero, z_end)

end do

view = ctop_views + 1 // Old Spectrum view

de_view_algorithm_{view} = de_algorithm_uniform

*de_view_halfwidth_{view} = const(3.) * PI / const(1.8, 2) // bigger*

```

vc_set(de_view_pointsview,de_view_start, r0ha, zero, z0ha)
z_end = zero // Use chord no. 1 from above set
r_end = r0ha + (z0ha - z_end) * tan(theta1 * PI / const(1.8, 2))
vc_set(de_view_pointsview,de_view_end, r_end, zero, z_end)

do view = 1, detector_total_views
  call detector_view_setup(vc_args(de_view_pointsview,de_view_start), de_view_halfwidthview,
    de_view_algorithmview, zone_frgs)
  do zone = 1, zn_num
    de_zone_frgszone,view = zone_frgszone
  end do
end do

var_free(zone_frgs)

num = ctop_views
var = de_var_unknown
tab_index = zero
var_min = zero
var_max = zero
mult = zero
spacing = de_spacing_unknown
do i = 1, num
  grp_viewsi = i
end do
call de_grp_init('Chord_integrals', num, var, tab_index, var_min, var_max, mult, spacing,
  grp_views)

num = 4 // Old: 1
var = de_var_wavelength
tab_index = 200
var_min = const(6558.)
var_max = const(6564.)
mult = const(1., -10)
spacing = de_spacing_linear
grp_views1 = 1 // Old: ctop_views+1
grp_views2 = 4
grp_views3 = 6
grp_views4 = 9
call de_grp_init('Halpha_spectrum', num, var, tab_index, var_min, var_max, mult, spacing,
  grp_views)
end if

return
end

```

Default detector setup. Just allocates arrays; no views are defined.

```
⟨ Null Detector Setup 1.17 ⟩ ≡  
  subroutine rg_detector_setup  
    implicit_none_f77  
    implicit_none_f90  
    call detector_setup  
    return  
  end
```

This code is used in section 1.12.

subroutine to read inputs from sonnet file.

⟨Functions and Subroutines 1.2⟩ +=

```

subroutine inpt_sonnet(nu, xmin, xmax, zmin, zmax, crx, cry, nx, ny, mx, my, ixcut1,
    ixcut2, iycut, null_type, ix_mirror)
    implicit_none_f77
    implicit_none_f90
    integer ix, iy, ixh, iyh, nxd, nyd, i, ixcut1, ixcut2, iycut, nu, mx, my, nx, ny, j, corner_no4,
        null_type, ix_mirror

    /* for /vesey/Sonnet/cmod.elm.779old */
    parameter (nxd = 120, nyd = 24)

    real crx(mx, my, 0 : 4), cry(mx, my, 0 : 4), pit(nxd, nyd), xmax, xmin, zmax, zmin, dummy1,
        fix_cornerx, fix_cornery

@#if 0
    real xcorner_old, ycorner_old, xcorner_new, ycorner_new
@#endif
    character dummy*80

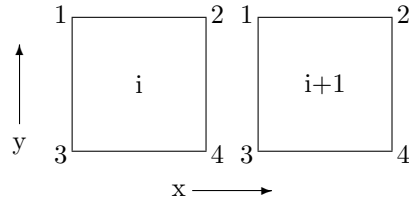
    xmax = 0.0 · 100D
    xmin = 1. · 105D
    zmax = 0.0 · 100D
    zmin = 1. · 105D
    nx = nxd
    ny = nyd    /* read coordinates */
    read (nu, 3366) dummy
    read (nu, 3366) dummy
    read (nu, 3366) dummy
    read (nu, 3366) dummy

    do iy = 1, nyd
        do ix = 1, nxd
            read (nu, 3333) ixh, iyh, crx(ix, iy, 1), cry(ix, iy, 1), crx(ix, iy, 2), cry(ix, iy, 2)
            read (nu, 3344) pit(ix, iy), dummy1, dummy1
            read (nu, 3333) ixh, iyh, crx(ix, iy, 3), cry(ix, iy, 3), crx(ix, iy, 4), cry(ix, iy, 4)
            read (nu, 3366) dummy

            /* get max and min of crx and cry */
            do i = 1, 4
                if (crx(ix, iy, i) > xmax)
                    xmax = crx(ix, iy, i)
                if (cry(ix, iy, i) > zmax)
                    zmax = cry(ix, iy, i)
                if (crx(ix, iy, i) < xmin)
                    xmin = crx(ix, iy, i)
                if (cry(ix, iy, i) < zmin)
                    zmin = cry(ix, iy, i)
            enddo
        enddo
    enddo

    /* the cuts may depend on the number u choose for the difference. i should probably take a
    relative measure. the configuration seems to be:

```



along the x direction ($iy=0$ along the lower edge of inner PFR) , we compare the bottom corners. these should be equal for adjacent cells until we reach the cut. this is $ixcut1$. from here as x increases, it traces the outer edge of the core and returns along the lower edge of the outer PFR. this is $ixcut2$. these two points are the x coordinates of the x-pt if u look at the whole thing in rectangular geometry. */

```

ixcut1 = 0
ixcut2 = 0
do i = 1, nxd - 1
  if ((abs(cry(i, 1, 4) - cry(i + 1, 1, 3)) ≥ 1. · 10-1)) then
    if (ixcut1 ≡ 0) then
      ixcut1 = i
    else
      ixcut2 = i
      goto 5
    endif
  endif
enddo
5: continue /* print *, 'ixcut1,2=', ixcut1,ixcut2 */

/* here we compare cells on either side of the cut in the y direction till we reach the x-point.
where one corner is common with the core. this gives the y coordinate of x point. */

do i = 1, nyd - 1
  if (abs(cry(ixcut1, i, 2) - cry(ixcut1 + 1, i, 1)) < 1. · 10-4) then
    iycut = i
    go to 6
  endif
enddo
6: continue /* print*, 'iycut', iycut */

corner_no1 = 3;
corner_no2 = 1;
corner_no3 = 2;
corner_no4 = 4

/* fix mismatch 6 for ix =30 in cmod.elm779old */

i = 30
do j = iycut + 1, nyd
  fix_cornerx = crx(i, j, 2)
  fix_cornery = cry(i, j, 2)
  call replace_corners(i + 1, j, 2, fix_cornerx, fix_cornery, corner_no, crx, cry, nx, ny, mx, my)
enddo

@#if 0
  // this is an example of how to exchange corners of two cells if required. here two corners ic1 and
  ic2 of [17,42] are exchanged.
  ic1 = 2

```

```

    ic2 = 3

    xcorner_old = crx17, 42, corner_noic1
    ycorner_old = cry17, 42, corner_noic1
    xcorner_new = crx17, 42, corner_noic2
    ycorner_new = cry17, 42, corner_noic2

    call replace_corners(17, 42, ic1, xcorner_new, ycorner_new, corner_no, crx, cry, nx, ny, mx, my)

    xcorner_new = xcorner_old
    ycorner_new = ycorner_old

    call replace_corners(17, 42, ic2, xcorner_new, ycorner_new, corner_no, crx, cry, nx, ny, mx, my)
@#endif

    call check_data(crx, cry, ixcut1, ixcut2, iycut, nxd, nyd, 3, 1, 2, 4, 3, 1, 2, 4, mx, my,
        null_type, ix_mirror)

@#if 0
    do i = 1, nxd
        do j = 1, nyd
            write(12, 3000) i, j, crx(i, j, 3), crx(i, j, 1), crx(i, j, 2), crx(i, j, 4), crx(i, j, 3)
            write(12, 3000) i, j, cry(i, j, 3), cry(i, j, 1), cry(i, j, 2), cry(i, j, 4), cry(i, j, 3)
        enddo
    enddo
@#endif

3000: format (2(i3, 2x), 5(e12.6, 2x))
3366: format (a)
3344: format (18x, e17.10, 14x, e17.10, 1x, e17.10) // 333 format(19x,i3,1x,i3,4x,e16.10,1x,e16.10)
3333: format (19x, i3, 1x, i3, 4x, e17.10, 1x, e17.10, 8x, e17.10, 1x, e17.10)

    return
end

```

replace corneri of cell ix,iy with [xcorner_new, ycorner_new].

⟨Functions and Subroutines 1.2⟩ +=

```

subroutine replace_corners(ix, iy, corneri, xcorner_new, ycorner_new, corner_no, x, y, nx, ny,
    mx, my)
    implicit_none_f77
    implicit_none_f90
    integer ix, iy, corneri, corner_no4, nx, ny, mx, my
    real xcorner_new, ycorner_new, x(mx, my, 0 : 4), y(mx, my, 0 : 4)

    if (corneri  $\equiv$  1) then
        xix, iy, corner_no1 = xcorner_new
        yix, iy, corner_no1 = ycorner_new
        if (iy  $\equiv$  1)
            goto 1
        xix, iy-1, corner_no2 = xcorner_new
        yix, iy-1, corner_no2 = ycorner_new
    1: if (ix  $\equiv$  1  $\vee$  iy  $\equiv$  1)
        goto 2
        xix-1, iy-1, corner_no3 = xcorner_new
        yix-1, iy-1, corner_no3 = ycorner_new
    2: if (ix  $\equiv$  1)
        goto 9
        xix-1, iy, corner_no4 = xcorner_new
        yix-1, iy, corner_no4 = ycorner_new
    elseif (corneri  $\equiv$  2) then
        xix, iy, corner_no2 = xcorner_new
        yix, iy, corner_no2 = ycorner_new
        if (ix  $\equiv$  1)
            goto 3
        xix-1, iy, corner_no3 = xcorner_new
        yix-1, iy, corner_no3 = ycorner_new
    3: if (ix  $\equiv$  1  $\vee$  iy  $\equiv$  ny)
        goto 4
        xix-1, iy+1, corner_no4 = xcorner_new
        yix-1, iy+1, corner_no4 = ycorner_new
    4: if (iy  $\equiv$  ny)
        goto 9
        xix, iy+1, corner_no1 = xcorner_new
        yix, iy+1, corner_no1 = ycorner_new
    elseif (corneri  $\equiv$  3) then
        xix, iy, corner_no3 = xcorner_new
        yix, iy, corner_no3 = ycorner_new
        if (iy  $\equiv$  ny)
            goto 5
        xix, iy+1, corner_no4 = xcorner_new
        yix, iy+1, corner_no4 = ycorner_new
    5: if (ix  $\equiv$  nx  $\vee$  iy  $\equiv$  ny)
        goto 6
        xix+1, iy+1, corner_no1 = xcorner_new
        yix+1, iy+1, corner_no1 = ycorner_new
    6: if (ix  $\equiv$  nx)

```

```

    goto 9
     $x_{ix+1}, iy, corner\_no_2 = xcorner\_new$ 
     $y_{ix+1}, iy, corner\_no_2 = ycorner\_new$ 
elseif ( $corneri \equiv 4$ ) then
     $x_{ix}, iy, corner\_no_4 = xcorner\_new$ 
     $y_{ix}, iy, corner\_no_4 = ycorner\_new$ 
    if ( $ix \equiv nx$ )
        goto 7
     $x_{ix+1}, iy, corner\_no_1 = xcorner\_new$ 
     $y_{ix+1}, iy, corner\_no_1 = ycorner\_new$ 
7: if ( $ix \equiv nx \vee iy \equiv 1$ )
    goto 8
     $x_{ix+1}, iy-1, corner\_no_2 = xcorner\_new$ 
     $y_{ix+1}, iy-1, corner\_no_2 = ycorner\_new$ 
8: if ( $iy \equiv 1$ )
    goto 9
     $x_{ix}, iy-1, corner\_no_3 = xcorner\_new$ 
     $y_{ix}, iy-1, corner\_no_3 = ycorner\_new$ 
endif
9: return
end

```

input routine for uedge data

(Functions and Subroutines 1.2) +≡

```

subroutine inpt_uedge(nu, nx, ny, ixpt1, ixpt2, iysptrx, rmu, zmu, m1x, m1y, null_type, ix_mirror)
    implicit none_f77
    implicit none_f90
    integer m1x, m1y // parameter (mx=120,my=50)
    real rmu_{m1x, m1y, 0:4}, zmu_{m1x, m1y, 0:4}
    integer ix, iy, i, nu, nx, ny, ixpt1, ixpt2, iysptrx, ix_mirror, null_type

    read (nu, *)
    read (nu, *) nx, ny
    read (nu, *) ixpt1, ixpt2
    read (nu, *) iysptrx
    read (nu, *) SP(((rmu(ix, iy, i), ix = 1, nx), iy = 1, ny), i = 0, 4)
    read (nu, *) SP(((zmu(ix, iy, i), ix = 1, nx), iy = 1, ny), i = 0, 4)
    /* its a good idea to check that data before going further */

    call check_data(rmu, zmu, ixpt1, ixpt2, iysptrx, nx, ny, 1, 3, 4, 2, 1, 3, 4, 2, m1x, m1y,
        null_type, ix_mirror)
    return
end

```

general routine to check consistency of data for degas2 geometry.

⟨Functions and Subroutines 1.2⟩ +=

```

subroutine check_data(x, y, ixcut1, ixcut2, iycut, nx, ny, xcorner1, xcorner2, xcorner3,
    xcorner4, ycorner1, ycorner2, ycorner3, ycorner4, m2x, m2y, null_type, ix_mirror)
    implicit_none_f77
    implicit_none_f90
    integer m2x, m2y
    real xm2x, m2y, 0:4, ym2x, m2y, 0:4 // input
    integer ix, iy, nx, ny, ixcut1, ixcut2, iycut, ix_mirror, null_type
    integer xcorner1, xcorner2, xcorner3, xcorner4, ycorner1, ycorner2, ycorner3, ycorner4
    logical single_null, double_null

    single_null = F
    double_null = F
    if (null_type ≡ 1) then
        single_null = T
    elseif (null_type ≡ 2) then
        double_null = T
        ix_mirror = nx / 2
    else
        print *, 'unknown_null_type'
        assert(F)
    endif

    /* —— check continuity of points —— check end points along x and top to bottom along y */
    if (ixcut1 > 0) then
        do ix = 1, ixcut1 - 1
            do iy = 1, iycut
                if (x(ix, iy, xcorner3) ≠ x(ix + 1, iy, xcorner2) ∨ x(ix, iy, xcorner4) ≠ x(ix + 1, iy,
                    xcorner1) ∨ y(ix, iy, ycorner3) ≠ y(ix + 1, iy, ycorner2) ∨ y(ix, iy,
                    ycorner4) ≠ y(ix + 1, iy, ycorner1)) then
                    print *, 'mismatch_1, ix, iy', ix, iy, ix + 1, iy
                    print *, 'corners_of_adjacent_cells_along_x_dont_match'
                    stop
                endif
                if (iy ≡ iycut)
                    goto 6
                if (x(ix, iy, xcorner2) ≠ x(ix, iy + 1, xcorner1) ∨ x(ix, iy, xcorner3) ≠ x(ix, iy + 1, xcorner4))
                    then
                        print *, 'mismatch_2, ix, iy', ix, iy, ix, iy + 1
                        print *, 'corners_of_top_of_lower_cell_and_bottom_of_upper_\
                            \cell_dont_match'
                        stop
                    endif
                6: continue
            enddo
        enddo
    end if // match end points along x and top to bottom along y from cut1 to cut2 for double nulls
        the mirror may come halfway in between, so dont check for continuity across it.

    do ix = max(1, ixcut1 + 1), ixcut2 - 1
        if (double_null ∧ ix ≡ ix_mirror)
            goto 70

```


2 INDEX

- abs: 1.18.
- assert: 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.21.
- b: [1.3](#).
- boltzmanns_const: 1.3, 1.11.
- boundaries_neighbors: 1.3.
- BOX: [1.2](#), 1.12.
- box_views: [1.13](#), 1.14.
- boxgen: 1.2, 1.13.
- check_data: 1.18, 1.20, [1.21](#).
- check_geometry: 1.3.
- command_arg: 1.2.
- complete: [1.3](#), 1.4.
- const: 1.3, 1.5, 1.11, 1.14, 1.16.
- corner_no: [1.18](#), [1.19](#).
- corneri: [1.19](#).
- crx: [1.18](#).
- cry: [1.18](#).
- ctop_views: [1.15](#), 1.16.
- current_material: [1.3](#), 1.4, 1.5, 1.7, 1.8, 1.9, 1.10.
- current_recyc_coef: [1.3](#), 1.4, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
- current_stratum: [1.3](#), 1.4, 1.5, 1.7, 1.9.
- current_temperature: [1.3](#), 1.4, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
- datafile_diskin: [1.3](#).
- datafilename: [1.3](#).
- dconst: 1.16.
- de_algorithm_uniform: 1.14, 1.16.
- de_common: 1.13, 1.14, 1.15, 1.16.
- de_grp_init: 1.14, 1.16.
- de_grps: 1.13, 1.15.
- de_spacing_linear: 1.14, 1.16.
- de_spacing_unknown: 1.14, 1.16.
- de_var_unknown: 1.14, 1.16.
- de_var_wavelength: 1.14, 1.16.
- de_view_algorithm: 1.13, 1.14, 1.15, 1.16.
- de_view_end: 1.14, 1.16.
- de_view_halfwidth: 1.13, 1.14, 1.15, 1.16.
- de_view_points: 1.13, 1.14, 1.15, 1.16.
- de_view_size: 1.13, 1.15.
- de_view_start: 1.14, 1.16.
- de_view_tab: 1.13, 1.15.
- de_zone_frags: 1.13, 1.14, 1.15, 1.16.
- de_zone_ind: 1.14, 1.16.
- declare_varp: 1.3, 1.14, 1.16.
- decompose_polygon: 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
- default_diag_setup: 1.3.
- define_dimen: 1.3.
- define_sector: 1.3.
- define_sector_exit: 1.3.
- define_sector_plasma: 1.3.
- define_sector_target: 1.3.
- define_sector_vacuum: 1.3.
- define_sector_wall: 1.3.
- define_varp: 1.3, 1.14, 1.16.
- degas: [1.3](#).
- DEGAS_CM0D: [1.2](#), 1.3, 1.12, 1.16.
- denehvt: 1.5.
- detector_setup: 1.17.
- detector_total_views: 1.13, 1.14, 1.15, 1.16.
- detector_view_setup: 1.14, 1.16.
- diag_grp_init: 1.3.
- diskin: 1.3.
- double_null: [1.3](#), 1.8, 1.9, 1.10, [1.21](#).
- dummy: [1.18](#).
- dummy1: [1.18](#).
- e: [1.3](#).
- electron_charge: 1.11.
- end_detectors: 1.3.
- end_sectors: 1.3.
- eof: 1.3.
- epsilon: 1.5.
- erase_geometry: 1.3.
- facearray: [1.3](#), 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
- face1: [1.3](#).
- file: 1.3.
- FILE: [1](#).
- FILELEN: 1.2, 1.3.
- filename: [1.2](#), [1.3](#).
- find_poly_zone: 1.3.
- fix_cornerx: [1.18](#).
- fix_cornery: [1.18](#).
- FLOAT: 1.3, 1.14, 1.16.
- form: 1.3.
- frabsorb: 1.4.
- GENERAL: [1.2](#).
- geometry: 1.
- GEOMETRY: [1.2](#), 1.3, 1.7, 1.12, 1.16.
- geometry_symmetry_cylindrical: 1.3.
- geometry_symmetry_oned: 1.3.
- geometry_symmetry_plane: 1.3.
- GEOMTRY: 1.2.
- gi_common: 1.3.
- gi_ext: 1.3.
- grid_sense: [1.3](#), 1.5, 1.7.
- grid_starth: [1.3](#), 1.5, 1.6.
- grid_startv: [1.3](#), 1.5, 1.6.
- grid_stoph: [1.3](#), 1.5, 1.6.

- grid_stopv*: [1.3](#), 1.5, 1.6.
gridx: 1.5, 1.6.
gridz: 1.5, 1.6.
grp_sector: 1.3.
grp_sectors1: [1.3](#).
grp_views: [1.14](#), [1.16](#).
h: [1.3](#).
half: 1.14, 1.16.
i: [1.3](#), [1.14](#), [1.16](#), [1.18](#), [1.20](#).
ic1: 1.18.
ic2: 1.18.
implicit_none_f77: 1.2, 1.3, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19, 1.20, 1.21.
implicit_none_f90: 1.2, 1.3, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19, 1.20, 1.21.
increment_num_polygon: [1.2](#), 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
init_geometry: 1.3, 1.7.
inpt: 1.3.
inpt_sonnet: 1.3, [1.18](#).
inpt_uedge: 1.3, [1.20](#).
INT: 1.3.
int_lookup: 1.3.
int_unused: 1.3.
integer: 1.1.
ix: [1.18](#), [1.19](#), [1.20](#), [1.21](#).
ix_mirror: [1.3](#), 1.8, 1.9, 1.10, [1.18](#), [1.20](#), [1.21](#).
ixcut1: [1.18](#), [1.21](#).
ixcut2: [1.18](#), [1.21](#).
ixh: [1.18](#).
ixpt1: [1.3](#), 1.8, 1.9, 1.10, [1.20](#).
ixpt2: [1.3](#), 1.8, 1.9, 1.10, 1.11, [1.20](#).
iy: [1.18](#), [1.19](#), [1.20](#), [1.21](#).
iycut: [1.18](#), [1.21](#).
iyh: [1.18](#).
iysp: [1.3](#).
iysptrx: 1.3, [1.20](#).
j: [1.3](#), [1.18](#).
j1: [1.3](#), 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
j2: [1.3](#), 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
keyword: [1.3](#).
kumat: 1.4.
kzone1: 1.5.
kzone2: 1.5.
len: 1.3.
length: [1.3](#).
line: [1.3](#), 1.4, 1.5, 1.6, 1.7.
LINELEN: 1.3.
lookup_surface: 1.3.
loop1: 1.3.
loop2: 1.3.
loop3: 1.3.
ma_check: 1.3.
ma_common: 1.3.
ma_lookup: 1.3, 1.4, 1.9, 1.11.
max: 1.21.
max_corner: 1.3, 1.7.
mem_delta: [1.2](#).
min_corner: 1.3, 1.7.
mindensity: [1.3](#), 1.5.
minx: [1.3](#), 1.5.
minz: [1.3](#), 1.5.
mod: 1.4.
mult: [1.3](#), [1.14](#), [1.16](#).
mx: [1.3](#), [1.18](#), [1.19](#).
my: [1.3](#), [1.18](#), [1.19](#).
m1x: [1.20](#).
m1y: [1.20](#).
m2x: [1.21](#).
m2y: [1.21](#).
n: [1.3](#).
namelist: 1.1.
nc_read_materials: 1.2.
new_zone_type: [1.3](#).
next_surface: [1.2](#), 1.3.
next_token: 1.3, 1.4, 1.5, 1.6, 1.7.
nosegsxz: 1.3, 1.4, 1.9.
nowals: 1.3.
nsectors: 1.3.
nu: [1.3](#), [1.18](#), [1.20](#).
null_type: [1.3](#), [1.18](#), [1.20](#), [1.21](#).
num: [1.3](#), 1.6, [1.14](#), [1.16](#).
num_points: [1.2](#), 1.3, 1.4.
num_polygon: 1.2, [1.3](#), 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11.
num_zone1: [1.3](#).
num_zone2: [1.3](#).
nx: [1.3](#), 1.7, 1.8, 1.9, 1.11, [1.18](#), [1.19](#), [1.20](#), [1.21](#).
nxd: [1.18](#).
ny: [1.3](#), 1.7, 1.8, 1.9, 1.11, [1.18](#), [1.19](#), [1.20](#), [1.21](#).
nyd: [1.18](#).
one: 1.3, 1.7, 1.14.
p: [1.3](#).
parse_string: 1.3.
pfr_recyc: [1.3](#), 1.11.
PI: 1.3, 1.7, 1.14, 1.16.
pit: [1.18](#).
plot: [1.3](#), 1.5.
points_ind: 1.3.
points_ind0: 1.3.

- poly_ind*: 1.3.
polygon_material: 1.2, 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
polygon_points: 1.2, 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
polygon_recyc_coef: 1.2, 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
polygon_segment: 1.2, 1.3, 1.4, 1.8, 1.9, 1.10, 1.11.
polygon_stratum: 1.2, 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
polygon_temperature: 1.2, 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
polygon_volume: 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
polygon_x: 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11.
polygon_zone: 1.2, 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
r_end: 1.16.
read_integer: 1.3, 1.4, 1.5, 1.6.
read_real: 1.3, 1.7.
read_string: 1.3.
readfilenames: 1.2.
readgeometry: 1.2.
readgeomfile: 1.2, 1.3.
real_undef: 1.3.
replace_corners: 1.18, 1.19.
rg_detector_setup: 1.3, 1.13, 1.15, 1.17.
rg_detector_setup_a: 1.13, 1.14, 1.15, 1.16.
rm: 1.3, 1.7, 1.8, 1.9, 1.10, 1.11.
rmu: 1.20.
r0ha: 1.16.

sc_check: 1.3.
sc_common: 1.3.
sc_diag_spacing_unknown: 1.3.
sc_diag_unknown: 1.3.
sect_zone1: 1.3.
sect_zone2: 1.3.
sector_surface: 1.3.
sector1: 1.3.
sector2: 1.3.
segment: 1.3.
segment1: 1.3.
set_zn_min_max: 1.3, 1.5, 1.7, 1.8, 1.9, 1.10, 1.11.
single_null: 1.3, 1.8, 1.9, 1.10, 1.21.
skip: 1.5.
sonnet: 1.3, 1.7.
SP: 1.3, 1.20.
spacing: 1.3, 1.14, 1.16.
st_decls: 1.3.
status: 1.3.
stderr: 1.3, 1.5.
stdout: 1.3, 1.5.

steph: 1.3, 1.6.
stepv: 1.3, 1.6.
stratum: 1.3.
symmetry: 1.3, 1.7.

tab_index: 1.3, 1.14, 1.16.
tan: 1.16.
target_recyc: 1.3, 1.11.
temp_material: 1.3, 1.4.
test_vec_1: 1.3, 1.7.
test_vec_2: 1.3, 1.7.
test_vec_3: 1.3, 1.7.
theta: 1.16.
trim: 1.3.
trx: 1.3.
twall: 1.4.
two: 1.3, 1.7.

uedge: 1.3, 1.7, 1.8, 1.9, 1.10.
UEDGE_RECT_SLAB: 1.2, 1.3, 1.7.
unit: 1.3.
universal_cell: 1.3, 1.7.

v: 1.3.
vacuum: 1.3, 1.7.
var: 1.3, 1.14, 1.16.
var_alloc: 1.13, 1.14, 1.15, 1.16.
var_free: 1.14, 1.16.
var_max: 1.3, 1.14, 1.16.
var_min: 1.3, 1.14, 1.16.
var_realloca: 1.2, 1.3.
vc_args: 1.14, 1.16.
vc_cross: 1.7.
vc_decl: 1.3.
vc_decls: 1.3.
vc_product: 1.7.
vc_set: 1.3, 1.5, 1.7, 1.14, 1.16.
view: 1.14, 1.16.
vol: 1.3, 1.7.
vol_test: 1.3, 1.5.
vola: 1.3.

wall_direction: 1.3, 1.4.
wall_no: 1.3, 1.4.
wall_recyc: 1.3, 1.11.
wall_start: 1.3, 1.4.
wall_stop: 1.3, 1.4.
wallfile: 1.7.
wallfile_diskin: 1.3.
wallfilename: 1.3.
web: 1.
WISING_CMOD: 1.2, 1.12, 1.16.
write_geometry: 1.3.

x: 1.19, 1.21.

x_test: [1.3](#), [1.5](#).
xcorner_new: [1.18](#), [1.19](#).
xcorner_old: [1.18](#).
xcorner1: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#), [1.21](#).
xcorner2: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.11](#), [1.21](#).
xcorner3: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.11](#), [1.21](#).
xcorner4: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#), [1.21](#).
xmax: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.11](#), [1.18](#).
xmin: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.11](#), [1.18](#).
xwall: [1.3](#), [1.4](#), [1.9](#).
xz_ind: [1.3](#).
xz_index: [1.2](#), [1.3](#).
x1: [1.3](#).
x2: [1.3](#).

y: [1.19](#), [1.21](#).
y_div: [1.3](#).
ycorner_new: [1.18](#), [1.19](#).
ycorner_old: [1.18](#).
ycorner1: [1.21](#).
ycorner2: [1.21](#).
ycorner3: [1.21](#).
ycorner4: [1.21](#).
yhat: [1.3](#), [1.7](#).
ymax: [1.3](#), [1.7](#).
ymin: [1.3](#), [1.7](#).

z_end: [1.16](#).
zcorner1: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zcorner2: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.11](#).
zcorner3: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.11](#).
zcorner4: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zero: [1.3](#), [1.4](#), [1.5](#), [1.7](#), [1.14](#), [1.16](#).
zi_ix: [1.3](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zi_iy: [1.3](#), [1.5](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zi_iz: [1.3](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zi_ptr: [1.3](#), [1.5](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zm: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zmax: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.11](#), [1.18](#).
zmin: [1.3](#), [1.7](#), [1.8](#), [1.9](#), [1.11](#), [1.18](#).
zmu: [1.20](#).
zn_common: [1.3](#), [1.13](#), [1.14](#), [1.15](#), [1.16](#).
zn_exit: [1.3](#), [1.10](#), [1.11](#).
zn_index: [1.3](#), [1.5](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zn_num: [1.3](#), [1.14](#), [1.16](#).
zn_plasma: [1.3](#), [1.5](#), [1.7](#).
zn_solid: [1.3](#), [1.8](#), [1.9](#), [1.11](#).
zn_type: [1.3](#).
zn_type_set: [1.3](#), [1.5](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zn_vacuum: [1.3](#), [1.9](#).
zn_volume: [1.3](#), [1.5](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zone: [1.3](#), [1.5](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#), [1.14](#), [1.16](#).
zone_center: [1.5](#), [1.7](#).
zone_fragments: [1.14](#), [1.16](#).
zonearray: [1.3](#), [1.5](#), [1.7](#), [1.8](#), [1.9](#), [1.10](#), [1.11](#).
zone1: [1.3](#).
zone2: [1.3](#).
zwall: [1.3](#), [1.4](#), [1.9](#).
z0ha: [1.16](#).

⟨Box Detector Setup 1.13, 1.14⟩ Used in section 1.12.
⟨Core Boundary Section 1.10⟩ Used in section 1.7.
⟨Edge Keyword 1.6⟩ Used in section 1.3.
⟨Functions and Subroutines 1.2, 1.3, 1.12, 1.18, 1.19, 1.20, 1.21⟩ Used in section 1.1.
⟨Grid Keyword 1.5⟩ Used in section 1.3.
⟨Memory allocation interface 0⟩ Used in sections 1.16, 1.15, 1.14, 1.13, and 1.3.
⟨Mesh Keyword 1.7⟩ Used in section 1.3.
⟨Not-Vacuum Boundary Section 1.8⟩ Used in section 1.7.
⟨Null Detector Setup 1.17⟩ Used in section 1.12.
⟨UERS Boundary Section 1.11⟩ Used in section 1.7.
⟨Vacuum Boundary Section 1.9⟩ Used in section 1.7.
⟨Wall Keyword 1.4⟩ Used in section 1.3.
⟨Wising C-Mod Detector Setup 1.15, 1.16⟩ Used in section 1.12.
⟨combal.h 0⟩ Used in section 1.3.
⟨comflg.h 0⟩ Used in section 1.3.
⟨comgeo.h 0⟩ Used in section 1.3.
⟨compar.h 0⟩ Used in section 1.3.
⟨compls.h 0⟩ Used in section 1.3.
⟨comrat.h 0⟩ Used in section 1.3.
⟨comrfl.h 0⟩ Used in section 1.3.
⟨comst.h 0⟩ Used in section 1.3.
⟨comstat.h 0⟩ Used in section 1.3.
⟨comsv.h 0⟩ Used in section 1.3.
⟨pardef.h 0⟩ Used in section 1.3.

COMMAND LINE: "fweave -f -i! -W[-ykw700 -ytw40000 -j -n/
/u/dstotler/degas2/src/readgeometry.web".

WEB FILE: "/u/dstotler/degas2/src/readgeometry.web".

CHANGE FILE: (none).

GLOBAL LANGUAGE: FORTRAN.