

Voro++

Generated by Doxygen 1.5.7

Sat Dec 13 23:28:57 2008

Contents

1	Voro++ class reference manual	2
1.1	Introduction	2
1.2	C++ class structure	2
1.3	The voronoicell class	2
1.3.1	Internal data representation	2
1.4	The container class	3
1.5	Wall computation	4
1.6	Extra functionality via the use of templates	5
2	Data Structure Documentation	6
2.1	container_base< r_option > Class Template Reference	6
2.1.1	Detailed Description	8
2.1.2	Constructor & Destructor Documentation	9
2.1.3	Member Function Documentation	9
2.1.4	Field Documentation	17
2.2	fatal_error Struct Reference	21
2.2.1	Detailed Description	22
2.2.2	Constructor & Destructor Documentation	22
2.3	neighbor_none Class Reference	22
2.3.1	Detailed Description	23
2.3.2	Constructor & Destructor Documentation	23
2.3.3	Member Function Documentation	23
2.4	neighbor_track Class Reference	26
2.4.1	Detailed Description	27
2.4.2	Constructor & Destructor Documentation	27
2.4.3	Member Function Documentation	28
2.4.4	Field Documentation	31
2.5	radius_mono Class Reference	32
2.5.1	Detailed Description	32
2.5.2	Constructor & Destructor Documentation	32
2.5.3	Member Function Documentation	32
2.5.4	Field Documentation	34
2.6	radius_poly Class Reference	35
2.6.1	Detailed Description	35

2.6.2	Constructor & Destructor Documentation	35
2.6.3	Member Function Documentation	35
2.6.4	Field Documentation	38
2.7	suretest Class Reference	38
2.7.1	Detailed Description	38
2.7.2	Constructor & Destructor Documentation	39
2.7.3	Member Function Documentation	39
2.7.4	Field Documentation	39
2.8	voronoicell_base< n_option > Class Template Reference	39
2.8.1	Detailed Description	41
2.8.2	Constructor & Destructor Documentation	41
2.8.3	Member Function Documentation	42
2.8.4	Field Documentation	50
2.9	voropp_loop Class Reference	51
2.9.1	Detailed Description	52
2.9.2	Constructor & Destructor Documentation	52
2.9.3	Member Function Documentation	52
2.9.4	Field Documentation	53
2.10	wall Class Reference	53
2.10.1	Detailed Description	53
2.10.2	Member Function Documentation	54
2.11	wall_cone Struct Reference	54
2.11.1	Detailed Description	55
2.11.2	Constructor & Destructor Documentation	55
2.11.3	Member Function Documentation	55
2.12	wall_cylinder Struct Reference	56
2.12.1	Detailed Description	57
2.12.2	Constructor & Destructor Documentation	57
2.12.3	Member Function Documentation	57
2.13	wall_plane Struct Reference	58
2.13.1	Detailed Description	59
2.13.2	Constructor & Destructor Documentation	59
2.13.3	Member Function Documentation	59
2.14	wall_sphere Struct Reference	60
2.14.1	Detailed Description	61

2.14.2	Constructor & Destructor Documentation	61
2.14.3	Member Function Documentation	61
3	File Documentation	62
3.1	cell.cc File Reference	62
3.1.1	Detailed Description	62
3.2	cell.hh File Reference	62
3.2.1	Detailed Description	63
3.2.2	Typedef Documentation	63
3.3	config.hh File Reference	63
3.3.1	Detailed Description	64
3.3.2	Define Documentation	64
3.3.3	Typedef Documentation	65
3.3.4	Variable Documentation	65
3.4	container.cc File Reference	67
3.4.1	Detailed Description	67
3.5	container.hh File Reference	67
3.5.1	Detailed Description	68
3.5.2	Typedef Documentation	68
3.6	voro++.cc File Reference	69
3.6.1	Detailed Description	69
3.7	voro++.hh File Reference	69
3.7.1	Detailed Description	69
3.8	wall.cc File Reference	69
3.8.1	Detailed Description	69
3.9	wall.hh File Reference	69
3.9.1	Detailed Description	70
3.10	worklist.cc File Reference	70
3.10.1	Detailed Description	70
3.11	worklist.hh File Reference	70
3.11.1	Detailed Description	70

1 Voro++ class reference manual

1.1 Introduction

Voro++ is a software library for carrying out 3D cell-based calculations of the Voronoi tessellation. It is primarily designed for applications in physics and materials science, where the Voronoi tessellation can be a useful tool in analyzing particle systems.

Voro++ is comprised of several C++ classes, and is designed to be incorporated into other programs. This manual provides a reference for every function in the class structure. For a general overview of the program, see the Voro++ website at <http://math.lbl.gov/voro++/> and in particular the example programs at <http://math.lbl.gov/voro++/examples/> that demonstrate many of the library's features.

1.2 C++ class structure

The code is structured around two main C++ classes. The `voronoicell` class contains all of the routines for constructing a single Voronoi cell. It represents the cell as a collection of vertices that are connected by edges, and there are routines for initializing, making, and outputting the cell. The container class represents a three-dimensional simulation region into which particles can be added. The class can then carry out a variety of Voronoi calculations by computing cells using the `voronoicell` class. It also has a general mechanism using virtual functions to implement walls, discussed in more detail below. To implement the radical Voronoi tessellation and the neighbor calculations, two class variants called `voronoicell_neighbor` and `container_poly` are provided by making use of templates, which is discussed below.

1.3 The `voronoicell` class

The `voronoicell` class represents a single Voronoi cell as a convex polyhedron, with a set of vertices that are connected by edges. The class contains a variety of functions that can be used to compute and output the Voronoi cell corresponding to a particular particle. The command `init()` can be used to initialize a cell as a large rectangular box. The Voronoi cell can then be computed by repeatedly cutting it with planes that correspond to the perpendicular bisectors between that particle and its neighbors.

This is achieved by using the `plane()` routine, which will recompute the cell's vertices and edges after cutting it with a single plane. This is the key routine in `voronoicell` class. It begins by exploiting the convexity of the underlying cell, tracing between edges to work out if the cell intersects the cutting plane. If it does not intersect, then the routine immediately exits. Otherwise, it finds an edge or vertex that intersects the plane, and from there, traces out a new face on the cell, recomputing the edge and vertex structure accordingly.

Once the cell is computed, it can be drawn using commands such as `draw_gnuplot()` and `draw_pov()`, or its volume can be evaluated using the `volume()` function. Many more routines are available, and are described in the online reference manual.

1.3.1 Internal data representation

The `voronoicell` class has a public member `p` representing the number of vertices. The polyhedral structure of the cell is stored in the following arrays:

- `pts[]`: an array of floating point numbers, that represent the position vectors $x_0, x_1, \dots, x_{\{p-1\}}$ of the polyhedron vertices.

- `nu[]`: the order of each vertex `n_0, n_1, ..., n_{p-1}`, corresponding to the number of other vertices to which each is connected.
- `ed[][]`: a table of edges and relations. For the *i*th vertex, `ed[i]` has $2n_i+1$ elements. The first n_i elements are the edges `e(j,i)`, where `e(j,i)` is the *j*th neighbor of vertex *i*. The edges are ordered according to a right-hand rule with respect to an outward-pointing normal. The next n_i elements are the relations `l(j,i)` which satisfy the property `e(l(j,i),e(j,i)) = i`. The final element of the `ed[i]` list is a back pointer used in memory allocation.

In a very large number of cases, the values of n_i will be 3. This is because the only way that a higher-order vertex can be created in the `plane()` routine is if the cutting plane perfectly intersects an existing vertex. For random particle arrangements with position vectors specified to double precision this should happen very rarely. A preliminary version of this code was quite successful with only making use of vertices of order 3. However, when calculating millions of cells, it was found that this approach is not robust, since a single floating point error can invalidate the computation. This can also be a problem for cases featuring crystalline arrangements of particles where the corresponding Voronoi cells may have high-order vertices by construction.

Because of this, Voro++ takes the approach that if an existing vertex is within a small numerical tolerance of the cutting plane, it is treated as being exactly on the plane, and the polyhedral topology is recomputed accordingly. However, while this improves robustness, it also adds the complexity that n_i may no longer always be 3. This causes memory management to be significantly more complicated, as different vertices require a different number of elements in the `ed[][]` array. To accommodate this, the `voronoi` class allocated edge memory in a different array called `mep[][]`, in such a way that all vertices of order *k* are held in `mep[k]`. If vertex *i* has order *k*, then `ed[i]` points to memory within `mep[k]`. The array `ed[][]` is never directly initialized as a two-dimensional array itself, but points at allocations within `mep[][]`. To the user, it appears as though each row of `ed[][]` has a different number of elements. When vertices are added or deleted, care must be taken to reorder and reassign elements in these arrays.

During the `plane()` routine, the code traces around the vertices of the cell, and adds new vertices along edges which intersect the cutting plane to create a new face. The values of `l(j,i)` are used in this computation, as when the code is traversing from one vertex on the cell to another, this information allows the code to immediately work out which edge of a vertex points back to the one it came from. As new vertices are created, the `l(j,i)` are also updated to ensure consistency. To ensure robustness, the plane cutting algorithm should work with any possible combination of vertices which are inside, outside, or exactly on the cutting plane.

Vertices exactly on the cutting plane create some additional computational difficulties. If there are two marginal vertices connected by an existing edge, then it would be possible for duplicate edges to be created between those two vertices, if the plane routine traces along both sides of this edge while constructing the new face. The code recognizes these cases and prevents the double edge from being formed. Another possibility is the formation of vertices of order two or one. At the end of the plane cutting routine, the code checks to see if any of these are present, removing the order one vertices by just deleting them, and removing the order two vertices by connecting the two neighbors of each vertex together. It is possible that the removal of a single low-order vertex could result in the creation of additional low-order vertices, so the process is applied recursively until no more are left.

1.4 The container class

The container class represents a three-dimensional rectangular box of particles. The constructor for this class sets up the coordinate ranges, sets whether each direction is periodic or not, and divides the box into a rectangular subgrid of regions. Particles can be added to the container using the `put()` command, that

adds a particle's position and an integer numerical ID label to the corresponding region. Alternatively, the command `import()` can be used to read large numbers of particles from a text file.

The key routine in this class is `compute_cell()`, which makes use of the `voronoicell` class to construct a Voronoi cell for a specific particle in the container. The basic approach that this function takes is to repeatedly cut the Voronoi cell by planes corresponding neighboring particles, and stop when it recognizes that all the remaining particles in the container are too far away to possibly influence cell's shape. The code makes use of two possible methods for working out when a cell computation is complete:

- Radius test: if the maximum distance of a Voronoi cell vertex from the cell center is R , then no particles more than a distance $2R$ away can possibly influence the cell. This a very fast computation to do, but it has no directionality: if the cell extends a long way in one direction then particles a long distance in other directions will still need to be tested.
- Region test: it is possible to test whether a specific region can possibly influence the cell by applying a series of plane tests at the point on the region which is closest to the Voronoi cell center. This is a slower computation to do, but it has directionality.

Another useful observation is that the regions that need to be tested are simply connected, meaning that if a particular region does not need to be tested, then neighboring regions which are further away do not need to be tested.

For maximum efficiency, it was found that a hybrid approach making use of both of the above tests worked well in practice. Radius tests work well for the first few blocks, but switching to region tests after then prevent the code from becoming extremely slow, due to testing over very large spherical shells of particles. The `compute_cell()` routine therefore takes the following approach:

- Initialize the `voronoicell` class to fill the entire computational domain.
- Cut the cell by any `wall` objects that have been added to the container.
- Apply plane cuts to the cell corresponding to the other particles which are within the current particle's region.
- Test over a pre-computed worklist of neighboring regions, that have been ordered according to the minimum distance away from the particle's position. Apply radius tests after every few regions to see if the calculation can terminate.
- If the code reaches the end of the worklist, add all the neighboring regions to a new list.
- Carry out a region test on the first item of the list. If the region needs to be tested, apply the `plane()` routine for all of its particles, and then add any neighboring regions to the end of the list that need to be tested. Continue until the list has no elements left.

The `compute_cell()` routine forms the basis of many other routines, such as `store_cell_volumes()` and `draw_cells_gnuplot()` that can be used to calculate and draw the cells in the entire container or in a subdomain.

1.5 Wall computation

Wall computations are handled by making use of a pure virtual `wall` class. Specific `wall` types are derived from this class, and require the specification of two routines: `point_inside()` that tests to see if a point is inside a `wall` or not, and `cut_cell()` that cuts a cell according to the wall's position. The walls can be added to the container using the `add_wall()` command, and these are called each time a `compute_cell()` command

is carried out. At present, [wall](#) types for planes, spheres, cylinders, and cones are provided, although custom walls can be added by creating new classes derived from the pure virtual class. Currently all [wall](#) types approximate the [wall](#) surface with a single plane, which produces some small errors, but generally gives good results for dense particle packings in direct contact with a [wall](#) surface. It would be possible to create more accurate walls by making `cut_cell()` routines that approximate the curved surface with multiple plane cuts.

The [wall](#) objects can be used for periodic calculations, although to obtain valid results, the walls should also be periodic as well. For example, in a domain that is periodic in the x direction, a cylinder aligned along the x axis could be added. At present, the interior of all [wall](#) objects are convex domains, and consequently any superposition of them will be a convex domain also. Carrying out computations in non-convex domains poses some problems, since this could theoretically lead to non-convex Voronoi cells, which the internal data representation of the `voronoi` class does not support. For non-convex cases where the [wall](#) surfaces feature just a small amount of negative curvature (eg. a torus) approximating the curved surface with a single plane cut may give an acceptable level of accuracy. For non-convex cases that feature internal angles, the best strategy may be to decompose the domain into several convex subdomains, carry out a calculation in each, and then add the results together. The `voronoi` class cannot be easily modified to handle non-convex cells as this would fundamentally alter the algorithms that it uses, and cases could arise where a single plane cut could create several new faces as opposed to just one.

1.6 Extra functionality via the use of templates

C++ templates are often presented as a mechanism for allowing functions to be coded to work with several different data types. However, they also provide an extremely powerful mechanism for achieving static polymorphism, allowing several variations of a program to be compiled from a single source code. `Voro++` makes use of templates in order to handle the radical Voronoi tessellation and the neighbor calculations, both of which require only relatively minimal alterations to the main body of code.

The main body of the `voronoi` class is written as a template named [voronoi_base](#). Two additional small classes are then written: [neighbor_track](#), which contains small, inlined functions that encapsulate all of the neighbor calculations, and [neighbor_none](#), which contains the same function names left blank. By making use of the `typedef` command, two classes are then created from the template:

- `voronoi`: an instance of [voronoi_base](#) with the [neighbor_none](#) class.
- `voronoi_neighbor`: an instance of [voronoi_base](#) with the [neighbor_track](#) class.

The two classes will be the same, except that the second will get all of the additional neighbor-tracking functionality compiled into it through the [neighbor_track](#) class. Since the two instances of the template are created during the compilation, and since all of the functions in [neighbor_none](#) and [neighbor_track](#) are inlined, there should be no speed overhead with this construction; it should have the same efficiency as writing two completely separate classes. C++ has other methods for achieving similar results, such as virtual functions and class inheritance, but these are more focused on dynamic polymorphism, switching between functionality at run-time, resulting in a drop in performance. This would be particularly apparent in this case, as the neighbor computation code, while small, is heavily integrated into the low-level details of the `plane()` routine, and a virtual function approach would require a very large number of function address look-ups.

In a similar manner, two small classes called [radius_mono](#) and [radius_poly](#) are provided. The first contains all routines suitable for calculating the standard Voronoi tessellation associated with a monodisperse particle packing, while the second incorporates variations to carry out the radical Voronoi tessellation associated with a polydisperse particle packing. Two classes are then created via `typedef` commands:

- container: an instance of `container_base` with the `radius_mono` class.
- container_poly: an instance of `container_base` with the `radius_poly` class.

The `container_poly` class accepts an additional variable in the `put()` command for the particle's radius. These radii are then used to weight the plane positions in the `compute_cell()` routine.

It should be noted that the underlying template structure is largely hidden from a typical user accessing the library's functionality, and as demonstrated in the examples, the classes listed above behave like regular C++ classes, and can be used in all the same ways. However, the template structure may provide an additional method of customizing the code; for example, an additional radius class could be written to implement a Voronoi tessellation variant.

2 Data Structure Documentation

2.1 `container_base< r_option >` Class Template Reference

A class representing the whole simulation region.

```
#include <container.hh>
```

Public Member Functions

- `container_base` (`fpoint` xa, `fpoint` xb, `fpoint` ya, `fpoint` yb, `fpoint` za, `fpoint` zb, int xn, int yn, int zn, bool xper, bool yper, bool zper, int memi)
- `~container_base` ()
- void `draw_particles` (const char *filename)
- void `draw_particles` ()
- void `draw_particles` (ostream &os)
- void `draw_particles_pov` (const char *filename)
- void `draw_particles_pov` ()
- void `draw_particles_pov` (ostream &os)
- void `import` (istream &is)
- void `import` ()
- void `import` (const char *filename)
- void `region_count` ()
- void `clear` ()
- void `draw_cells_gnuplot` (const char *filename, `fpoint` xmin, `fpoint` xmax, `fpoint` ymin, `fpoint` ymax, `fpoint` zmin, `fpoint` zmax)
- void `draw_cells_gnuplot` (const char *filename)
- void `draw_cells_pov` (const char *filename, `fpoint` xmin, `fpoint` xmax, `fpoint` ymin, `fpoint` ymax, `fpoint` zmin, `fpoint` zmax)
- void `draw_cells_pov` (const char *filename)
- void `store_cell_volumes` (`fpoint` *bb)
- `fpoint` `packing_fraction` (`fpoint` *bb, `fpoint` cx, `fpoint` cy, `fpoint` cz, `fpoint` r)
- `fpoint` `packing_fraction` (`fpoint` *bb, `fpoint` xmin, `fpoint` xmax, `fpoint` ymin, `fpoint` ymax, `fpoint` zmin, `fpoint` zmax)
- `fpoint` `sum_cell_volumes` ()
- void `compute_all_cells` ()

- void `count_all_faces` (ostream &os)
- void `count_all_faces` ()
- void `count_all_faces` (const char *filename)
- void `print_all` (ostream &os)
- void `print_all` ()
- void `print_all` (const char *filename)
- void `print_all_neighbor` (ostream &os)
- void `print_all_neighbor` ()
- void `print_all_neighbor` (const char *filename)
- template<class n_option >
bool `compute_cell_sphere` (`voronoicell_base`< n_option > &c, int i, int j, int k, int ijk, int s)
- template<class n_option >
bool `compute_cell_sphere` (`voronoicell_base`< n_option > &c, int i, int j, int k, int ijk, int s, `fpoint` x, `fpoint` y, `fpoint` z)
- template<class n_option >
bool `compute_cell` (`voronoicell_base`< n_option > &c, int i, int j, int k, int ijk, int s)
- template<class n_option >
bool `compute_cell` (`voronoicell_base`< n_option > &c, int i, int j, int k, int ijk, int s, `fpoint` x, `fpoint` y, `fpoint` z)
- void `put` (int n, `fpoint` x, `fpoint` y, `fpoint` z)
- void `put` (int n, `fpoint` x, `fpoint` y, `fpoint` z, `fpoint` r)
- void `add_wall` (wall &w)
- bool `point_inside` (`fpoint` x, `fpoint` y, `fpoint` z)
- bool `point_inside_walls` (`fpoint` x, `fpoint` y, `fpoint` z)

Protected Member Functions

- template<class n_option >
void `print_all` (ostream &os, `voronoicell_base`< n_option > &c)
- template<class n_option >
bool `initialize_voronoicell` (`voronoicell_base`< n_option > &c, `fpoint` x, `fpoint` y, `fpoint` z)
- void `add_particle_memory` (int i)
- void `add_list_memory` ()

Protected Attributes

- const `fpoint` ax
- const `fpoint` bx
- const `fpoint` ay
- const `fpoint` by
- const `fpoint` az
- const `fpoint` bz
- const `fpoint` xsp
- const `fpoint` ysp
- const `fpoint` zsp
- const int nx
- const int ny
- const int nz

- const int `nxy`
- const int `nxyz`
- const int `hx`
- const int `hy`
- const int `hz`
- const int `hxy`
- const int `hxyz`
- const bool `xperiodic`
- const bool `yperiodic`
- const bool `zperiodic`
- int * `co`
- int * `mem`
- int ** `id`
- unsigned int * `mask`
- int * `sl`
- unsigned int `mv`
- int `s_start`
- int `s_end`
- int `s_size`
- `fpoint` ** `p`
- `wall` ** `walls`
- int `wall_number`
- int `current_wall_size`
- `r_option` `radius`
- int `sz`
- `fpoint` * `mrاد`

Friends

- class `voropp_loop`
- class `radius_poly`

2.1.1 Detailed Description

template<class `r_option`> class `container_base`< `r_option` >

A class representing the whole simulation region.

The container class represents the whole simulation region. The container constructor sets up the geometry and periodicity, and divides the geometry into rectangular grid of blocks, each of which handles the particles in a particular area. Routines exist for putting in particles, importing particles from standard input, and carrying out Voronoi calculations.

Definition at line 33 of file `container.hh`.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 `template<class r_option > container_base< r_option >::container_base (fpoint xa, fpoint xb, fpoint ya, fpoint yb, fpoint za, fpoint zb, int xn, int yn, int zn, bool xper, bool yper, bool zper, int memi)` [inline]

Container constructor. The first six arguments set the corners of the box to be (*xa,ya,za*) and (*xb,yb,zb*). The box is then divided into an *nx* by *ny* by *nz* grid of blocks, set by the following three arguments. The next three arguments are booleans, which set the periodicity in each direction. The final argument sets the amount of memory allocated to each block.

Definition at line 20 of file container.cc.

2.1.2.2 `template<class r_option > container_base< r_option >::~~container_base ()` [inline]

Container destructor - free memory.

Definition at line 135 of file container.cc.

2.1.3 Member Function Documentation

2.1.3.1 `template<class r_option > void container_base< r_option >::add_list_memory ()` [inline, protected]

Add list memory.

Definition at line 268 of file container.cc.

2.1.3.2 `template<class r_option > void container_base< r_option >::add_particle_memory (int i)` [inline, protected]

Increase memory for a particular region.

Definition at line 250 of file container.cc.

2.1.3.3 `template<class r_option > void container_base< r_option >::add_wall (wall & w)` [inline]

Adds a [wall](#) to the container.

Parameters:

← *&w* a [wall](#) object to be added.

Definition at line 1411 of file container.cc.

2.1.3.4 `template<class r_option > void container_base< r_option >::clear ()` [inline]

Clears a container of particles.

Definition at line 322 of file container.cc.

2.1.3.5 `template<class r_option > void container_base< r_option >::compute_all_cells ()` [inline]

This function computes all the cells in the container, but does nothing with the output. It is useful for measuring the pure computation time of the Voronoi algorithm, without any extraneous calculations, such as volume evaluation or cell output.

Definition at line 393 of file container.cc.

2.1.3.6 `template<class r_option > template<class n_option > bool container_base< r_option >::compute_cell (voronoicell_base< n_option > & c, int i, int j, int k, int ijk, int s, fpoint x, fpoint y, fpoint z)` [inline]

This routine computes a Voronoi cell for a single particle in the container. It can be called by the user, but is also forms the core part of several of the main functions, such as [store_cell_volumes\(\)](#), [print_all\(\)](#), and the drawing routines. The algorithm constructs the cell by testing over the neighbors of the particle, working outwards until it reaches those particles which could not possibly intersect the cell. For maximum efficiency, this algorithm is divided into three parts. In the first section, the algorithm tests over the blocks which are in the immediate vicinity of the particle, by making use of one of the precomputed worklists. The code then continues to test blocks on the worklist, but also begins to construct a list of neighboring blocks outside the worklist which may need to be test. In the third section, the routine starts testing these neighboring blocks, evaluating whether or not a particle in them could possibly intersect the cell. For blocks that intersect the cell, it tests the particles in that block, and then adds the block neighbors to the list of potential places to consider.

Parameters:

- ← *&c* a reference to a voronoicell object.
- ← *(i,j,k)* the coordinates of the block that the test particle is in.
- ← *ijk* the index of the block that the test particle is in, set to $i+nx*(j+ny*k)$.
- ← *s* the index of the particle within the test block.
- ← *(x,y,z)* The coordinates of the particle.
- ← *s* the index of the particle within the test block.

Definition at line 745 of file container.cc.

2.1.3.7 `template<class r_option > template<class n_option > bool container_base< r_option >::compute_cell (voronoicell_base< n_option > & c, int i, int j, int k, int ijk, int s)` [inline]

An overloaded version of `compute_cell`, that sets up the *x*, *y*, and *z* variables. It can be run by the user, and it is also called multiple times by the functions [print_all\(\)](#), [store_cell_volumes\(\)](#), and the output routines.

Parameters:

- ← *&c* a reference to a voronoicell object.
- ← *(i,j,k)* the coordinates of the block that the test particle is in.
- ← *ijk* the index of the block that the test particle is in, set to $i+nx*(j+ny*k)$.
- ← *s* the index of the particle within the test block.

Definition at line 714 of file container.cc.

2.1.3.8 `template<class r_option > template<class n_option > bool container_base< r_option >::compute_cell_sphere (voronoicell_base< n_option > & c, int i, int j, int k, int ijk, int s, fpoint x, fpoint y, fpoint z) [inline]`

This routine is a simpler alternative to `compute_cell()`, that constructs the cell by testing over successively larger spherical shells of particles. For a container that is homogeneously filled with particles, this routine runs as fast as `compute_cell()`. However, it rapidly becomes inefficient for cases when the particles are not homogeneously distributed, or where parts of the container might not be filled. In that case, the spheres may grow very large before being cut off, leading to poor efficiency.

Parameters:

- ← *&c* a reference to a voronoicell object.
- ← *(i,j,k)* the coordinates of the block that the test particle is in.
- ← *ijk* the index of the block that the test particle is in, set to $i+nx*(j+ny*k)$.
- ← *s* the index of the particle within the test block.
- ← *(x,y,z)* The coordinates of the particle.

Definition at line 657 of file container.cc.

2.1.3.9 `template<class r_option > template<class n_option > bool container_base< r_option >::compute_cell_sphere (voronoicell_base< n_option > & c, int i, int j, int k, int ijk, int s) [inline]`

An overloaded version of `compute_cell_sphere()`, that sets up the x, y, and z variables.

Parameters:

- ← *&c* a reference to a voronoicell object.
- ← *(i,j,k)* the coordinates of the block that the test particle is in.
- ← *ijk* the index of the block that the test particle is in, set to $i+nx*(j+ny*k)$.
- ← *s* the index of the particle within the test block.

Definition at line 698 of file container.cc.

2.1.3.10 `template<class r_option > void container_base< r_option >::count_all_faces (const char * filename) [inline]`

An overloaded version of `count_all_faces()`, which outputs the result to a particular file.

Parameters:

- ← *filename* The name of the file to write to.

Definition at line 515 of file container.cc.

2.1.3.11 `template<class r_option > void container_base< r_option >::count_all_faces () [inline]`

Prints a list of all particle labels, positions, and the number of faces to the standard output.

Definition at line 507 of file container.cc.

2.1.3.12 `template<class r_option > void container_base< r_option >::count_all_faces (ostream & os)`
[inline]

Prints a list of all particle labels, positions, and the number of faces to the standard output.

Definition at line 488 of file container.cc.

2.1.3.13 `template<class r_option > void container_base< r_option >::draw_cells_gnuplot (const char * filename)` [inline]

If only a filename is supplied to [draw_cells_gnuplot\(\)](#), then assume that we are calculating the entire simulation region.

Definition at line 353 of file container.cc.

2.1.3.14 `template<class r_option > void container_base< r_option >::draw_cells_gnuplot (const char * filename, fpoint xmin, fpoint xmax, fpoint ymin, fpoint ymax, fpoint zmin, fpoint zmax)` [inline]

Computes the Voronoi cells for all particles within a box with corners (xmin,ymin,zmin) and (xmax,ymax,zmax), and saves the output in a format that can be read by gnuplot.

Definition at line 331 of file container.cc.

2.1.3.15 `template<class r_option > void container_base< r_option >::draw_cells_pov (const char * filename)` [inline]

If only a filename is supplied to [draw_cells_pov\(\)](#), then assume that we are calculating the entire simulation region.

Definition at line 384 of file container.cc.

2.1.3.16 `template<class r_option > void container_base< r_option >::draw_cells_pov (const char * filename, fpoint xmin, fpoint xmax, fpoint ymin, fpoint ymax, fpoint zmin, fpoint zmax)` [inline]

Computes the Voronoi cells for all particles within a box with corners (xmin,ymin,zmin) and (xmax,ymax,zmax), and saves the output in a format that can be read by gnuplot.

Definition at line 361 of file container.cc.

2.1.3.17 `template<class r_option > void container_base< r_option >::draw_particles (ostream & os)`
[inline]

Dumps all the particle positions and identifies to a file.

Definition at line 151 of file container.cc.

2.1.3.18 `template<class r_option > void container_base< r_option >::draw_particles ()` [inline]

An overloaded version of the [draw_particles\(\)](#) routine, that just prints to standard output.

Definition at line 165 of file container.cc.

2.1.3.19 `template<class r_option > void container_base< r_option >::draw_particles (const char * filename) [inline]`

An overloaded version of the [draw_particles\(\)](#) routine, that outputs the particle positions to a file.

Parameters:

← *filename* the file to write to.

Definition at line 173 of file container.cc.

2.1.3.20 `template<class r_option > void container_base< r_option >::draw_particles_pov (ostream & os) [inline]`

Dumps all the particle positions in the POV-Ray format.

Definition at line 182 of file container.cc.

2.1.3.21 `template<class r_option > void container_base< r_option >::draw_particles_pov () [inline]`

An overloaded version of the [draw_particles_pov\(\)](#) routine, that just prints to standard output.

Definition at line 198 of file container.cc.

2.1.3.22 `template<class r_option > void container_base< r_option >::draw_particles_pov (const char * filename) [inline]`

An overloaded version of the [draw_particles_pov\(\)](#) routine, that outputs the particle positions to a file.

Parameters:

← *filename* the file to write to.

Definition at line 206 of file container.cc.

2.1.3.23 `template<class r_option > void container_base< r_option >::import (const char * filename) [inline]`

An overloaded version of the import routine, that reads in particles from a particular file.

Parameters:

← *filename* The name of the file to read from.

Definition at line 302 of file container.cc.

2.1.3.24 `template<class r_option > void container_base< r_option >::import () [inline]`

An overloaded version of the import routine, that reads the standard input.

Definition at line 294 of file container.cc.

2.1.3.25 `template<class r_option > void container_base< r_option >::import (istream & is) [inline]`

Import a list of particles from standard input.

Definition at line 287 of file container.cc.

2.1.3.26 `template<class r_option > template<class n_option > bool container_base< r_option >::initialize_voronocell (voronocell_base< n_option > & c, fpoint x, fpoint y, fpoint z) [inline, protected]`

Initialize the Voronoi cell to be the entire container. For non-periodic coordinates, this is set by the position of the walls. For periodic coordinates, the space is equally divided in either direction from the particle's initial position. That makes sense since those boundaries would be made by the neighboring periodic images of this particle.

Definition at line 606 of file container.cc.

2.1.3.27 `template<class r_option > fpoint container_base< r_option >::packing_fraction (fpoint * bb, fpoint xmin, fpoint xmax, fpoint ymin, fpoint ymax, fpoint zmin, fpoint zmax) [inline]`

Computes the local packing fraction at a point, by summing the volumes of all particles within test box, and dividing by the sum of their Voronoi volumes that were previous computed using the [store_cell_volumes\(\)](#) function.

Parameters:

- ← **bb* an array holding the Voronoi volumes of the particles.
- ← (*xmin,ymin,zmin*) the minimum coordinates of the box.
- ← (*xmax,ymax,zmax*) the maximum coordinates of the box.

Definition at line 454 of file container.cc.

2.1.3.28 `template<class r_option > fpoint container_base< r_option >::packing_fraction (fpoint * bb, fpoint cx, fpoint cy, fpoint cz, fpoint r) [inline]`

Computes the local packing fraction at a point, by summing the volumes of all particles within a test sphere, and dividing by the sum of their Voronoi volumes that were previous computed using the [store_cell_volumes\(\)](#) function.

Parameters:

- ← **bb* an array holding the Voronoi volumes of the particles.
- ← (*cx,cy,cz*) the center of the test sphere.
- ← *r* the radius of the test sphere.

Definition at line 427 of file container.cc.

2.1.3.29 `template<class r_option > bool container_base< r_option >::point_inside (fpoint x, fpoint y, fpoint z) [inline]`

This function tests to see if a given vector lies within the container bounds and any walls.

Parameters:

← (x,y,z) The position vector to be tested.

Returns:

True if the point is inside the container, false if the point is outside.

Definition at line 624 of file container.cc.

2.1.3.30 `template<class r_option > bool container_base< r_option >::point_inside_walls (fpoint x, fpoint y, fpoint z) [inline]`

This function tests to see if a give vector lies within the walls that have been added to the container, but does not specifically check whether the vector lies within the container bounds.

Parameters:

← (x,y,z) The position vector to be tested.

Returns:

True if the point is inside the container, false if the point is outside.

Definition at line 636 of file container.cc.

2.1.3.31 `template<class r_option > template<class n_option > void container_base< r_option >::print_all (ostream & os, voronoicell_base< n_option > & c) [inline, protected]`

Prints a list of all particle labels, positions, and Voronoi volumes to the standard output.

Definition at line 526 of file container.cc.

2.1.3.32 `template<class r_option > void container_base< r_option >::print_all (const char * filename) [inline]`

An overloaded version of [print_all\(\)](#), which outputs the result to a particular file.

Parameters:

← *filename* The name of the file to write to.

Definition at line 562 of file container.cc.

2.1.3.33 `template<class r_option > void container_base< r_option >::print_all () [inline]`

An overloaded version of [print_all\(\)](#), which just prints to standard output.

Definition at line 553 of file container.cc.

2.1.3.34 `template<class r_option > void container_base< r_option >::print_all (ostream & os) [inline]`

Prints a list of all particle labels, positions, and Voronoi volumes to the standard output.

Definition at line 546 of file container.cc.

2.1.3.35 `template<class r_option > void container_base< r_option >::print_all_neighbor (const char * filename) [inline]`

An overloaded version of `print_all_neighbor()`, which outputs the result to a particular file

Parameters:

← *filename* The name of the file to write to.

Definition at line 591 of file container.cc.

2.1.3.36 `template<class r_option > void container_base< r_option >::print_all_neighbor () [inline]`

An overloaded version of `print_all_neighbor()`, which just prints to standard output.

Definition at line 582 of file container.cc.

2.1.3.37 `template<class r_option > void container_base< r_option >::print_all_neighbor (ostream & os) [inline]`

Prints a list of all particle labels, positions, Voronoi volumes, and a list of neighboring particles to an output stream.

Parameters:

← *os* The output stream to print to.

Definition at line 574 of file container.cc.

2.1.3.38 `template<class r_option > void container_base< r_option >::put (int n, fpoint x, fpoint y, fpoint z, fpoint r) [inline]`

Put a particle into the correct region of the container.

Parameters:

- ← *n* The numerical ID of the inserted particle.
- ← (*x,y,z*) The position vector of the inserted particle.
- ← *r* The radius of the particle.

Definition at line 234 of file container.cc.

2.1.3.39 `template<class r_option > void container_base< r_option >::put (int n, fpoint x, fpoint y, fpoint z) [inline]`

Put a particle into the correct region of the container.

Definition at line 215 of file container.cc.

2.1.3.40 `template<class r_option > void container_base< r_option >::region_count () [inline]`

Outputs the number of particles within each region.

Definition at line 311 of file container.cc.

2.1.3.41 `template<class r_option > void container_base< r_option >::store_cell_volumes (fpoint * bb)`
[inline]

Computes the Voronoi volumes for all the particles, and stores the results according to the particle label in the fpoint array bb.

Definition at line 405 of file container.cc.

2.1.3.42 `template<class r_option > fpoint container_base< r_option >::sum_cell_volumes ()`
[inline]

Computes the Voronoi volumes for all the particles, and stores the results according to the particle label in the fpoint array bb.

Definition at line 476 of file container.cc.

2.1.4 Field Documentation

2.1.4.1 `template<class r_option > const fpoint container_base< r_option >::ax` [protected]

The minimum x coordinate of the container.

Definition at line 81 of file container.hh.

2.1.4.2 `template<class r_option > const fpoint container_base< r_option >::ay` [protected]

The minimum y coordinate of the container.

Definition at line 85 of file container.hh.

2.1.4.3 `template<class r_option > const fpoint container_base< r_option >::az` [protected]

The minimum z coordinate of the container.

Definition at line 89 of file container.hh.

2.1.4.4 `template<class r_option > const fpoint container_base< r_option >::bx` [protected]

The maximum x coordinate of the container.

Definition at line 83 of file container.hh.

2.1.4.5 `template<class r_option > const fpoint container_base< r_option >::by` [protected]

The maximum y coordinate of the container.

Definition at line 87 of file container.hh.

2.1.4.6 `template<class r_option > const fpoint container_base< r_option >::bz` [protected]

The maximum z coordinate of the container.

Definition at line 91 of file container.hh.

2.1.4.7 `template<class r_option > int* container_base< r_option >::co` [protected]

This array holds the number of particles within each computational box of the container.

Definition at line 138 of file container.hh.

2.1.4.8 `template<class r_option > int container_base< r_option >::current_wall_size` [protected]

The current amount of memory allocated for walls.

Definition at line 174 of file container.hh.

2.1.4.9 `template<class r_option > const int container_base< r_option >::hx` [protected]

The number of boxes in the x direction for the searching mask.

Definition at line 115 of file container.hh.

2.1.4.10 `template<class r_option > const int container_base< r_option >::hxy` [protected]

A constant, set to the value of hx multiplied by hy, which is used in the routines which step through mask boxes in sequence.

Definition at line 123 of file container.hh.

2.1.4.11 `template<class r_option > const int container_base< r_option >::hxyz` [protected]

A constant, set to the value of hx*hy*hz, which is used in the routines which step through mask boxes in sequence.

Definition at line 126 of file container.hh.

2.1.4.12 `template<class r_option > const int container_base< r_option >::hy` [protected]

The number of boxes in the y direction for the searching mask.

Definition at line 117 of file container.hh.

2.1.4.13 `template<class r_option > const int container_base< r_option >::hz` [protected]

The number of boxes in the z direction for the searching mask.

Definition at line 119 of file container.hh.

2.1.4.14 `template<class r_option > int container_base< r_option >::id` [protected]**

This array holds the numerical IDs of each particle in each computational box.

Definition at line 147 of file container.hh.

2.1.4.15 `template<class r_option > unsigned int* container_base< r_option >::mask` [protected]

This array is used as a mask.

Definition at line 149 of file container.hh.

2.1.4.16 `template<class r_option > int* container_base< r_option >::mem` [protected]

This array holds the maximum amount of particle memory for each computational box of the container. If the number of particles in a particular box ever approaches this limit, more is allocated using the [add_particle_memory\(\)](#) function.

Definition at line 144 of file container.hh.

2.1.4.17 `template<class r_option > fpoint* container_base< r_option >::mrad` [protected]

An array to hold the minimum distances associated with the worklists. This array is initialized during container construction, by the `initialize_radii()` routine.

Definition at line 195 of file container.hh.

2.1.4.18 `template<class r_option > unsigned int container_base< r_option >::mv` [protected]

This sets the current value being used to mark tested blocks in the mask.

Definition at line 155 of file container.hh.

2.1.4.19 `template<class r_option > const int container_base< r_option >::nx` [protected]

The number of boxes in the x direction.

Definition at line 102 of file container.hh.

2.1.4.20 `template<class r_option > const int container_base< r_option >::nxy` [protected]

A constant, set to the value of `nx` multiplied by `ny`, which is used in the routines which step through boxes in sequence.

Definition at line 110 of file container.hh.

2.1.4.21 `template<class r_option > const int container_base< r_option >::nxyz` [protected]

A constant, set to the value of `nx*ny*nz`, which is used in the routines which step through boxes in sequence.

Definition at line 113 of file container.hh.

2.1.4.22 `template<class r_option > const int container_base< r_option >::ny` [protected]

The number of boxes in the y direction.

Definition at line 104 of file container.hh.

2.1.4.23 `template<class r_option > const int container_base< r_option >::nz` [protected]

The number of boxes in the z direction.

Definition at line 106 of file container.hh.

2.1.4.24 `template<class r_option > fpoint** container_base< r_option >::p` [protected]

A two dimensional array holding particle positions. For the derived container_poly class, this also holds particle radii.

Definition at line 167 of file container.hh.

2.1.4.25 `template<class r_option > r_option container_base< r_option >::radius` [protected]

This object contains all the functions for handling how the particle radii should be treated. If the template is instantiated with the [radius_mono](#) class, then this object contains mostly blank routines that do nothing to the cell computation, to compute the basic Voronoi diagram. If the template is instantiated with the [radius_poly](#) calls, then this object provides routines for modifying the Voronoi cell computation in order to create the radical Voronoi tessellation.

Definition at line 184 of file container.hh.

2.1.4.26 `template<class r_option > int container_base< r_option >::s_end` [protected]

The position of the last element on the search list to be considered.

Definition at line 161 of file container.hh.

2.1.4.27 `template<class r_option > int container_base< r_option >::s_size` [protected]

The current size of the search list.

Definition at line 163 of file container.hh.

2.1.4.28 `template<class r_option > int container_base< r_option >::s_start` [protected]

The position of the first element on the search list to be considered.

Definition at line 158 of file container.hh.

2.1.4.29 `template<class r_option > int* container_base< r_option >::sl` [protected]

This array is used to store the list of blocks to test during the Voronoi cell computation.

Definition at line 152 of file container.hh.

2.1.4.30 `template<class r_option > int container_base< r_option >::sz` [protected]

The amount of memory in the array structure for each particle. This is set to 3 when the basic class is initialized, so that the array holds (x,y,z) positions. If the container class is initialized as part of the derived class container_poly, then this is set to 4, to also hold the particle radii.

Definition at line 191 of file container.hh.

2.1.4.31 `template<class r_option > int container_base< r_option >::wall_number` [protected]

The current number of [wall](#) objects, initially set to zero.

Definition at line 172 of file container.hh.

2.1.4.32 `template<class r_option > wall** container_base< r_option >::walls` [protected]

This array holds pointers to any [wall](#) objects that have been added to the container.

Definition at line 170 of file container.hh.

2.1.4.33 `template<class r_option > const bool container_base< r_option >::xperiodic` [protected]

A boolean value that determines if the x coordinate is periodic or not.

Definition at line 129 of file container.hh.

2.1.4.34 `template<class r_option > const fpoint container_base< r_option >::xsp` [protected]

The inverse box length in the x direction, set to $n_x/(b_x-a_x)$.

Definition at line 94 of file container.hh.

2.1.4.35 `template<class r_option > const bool container_base< r_option >::yperiodic` [protected]

A boolean value that determines if the y coordinate is periodic or not.

Definition at line 132 of file container.hh.

2.1.4.36 `template<class r_option > const fpoint container_base< r_option >::ysp` [protected]

The inverse box length in the y direction, set to $n_y/(b_y-a_y)$.

Definition at line 97 of file container.hh.

2.1.4.37 `template<class r_option > const bool container_base< r_option >::zperiodic` [protected]

A boolean value that determines if the z coordinate is periodic or not.

Definition at line 135 of file container.hh.

2.1.4.38 `template<class r_option > const fpoint container_base< r_option >::zsp` [protected]

The inverse box length in the z direction, set to $n_z/(b_z-a_z)$.

Definition at line 100 of file container.hh.

The documentation for this class was generated from the following files:

- [container.hh](#)
- [container.cc](#)
- [worklist.cc](#)

2.2 fatal_error Struct Reference

Structure for printing fatal error messages and exiting.


```
#include <cell.hh>
```

Public Member Functions

- [fatal_error](#) (const char *p)

2.2.1 Detailed Description

Structure for printing fatal error messages and exiting.

Structure for printing fatal error messages and exiting.

Definition at line 23 of file cell.hh.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 fatal_error::fatal_error (const char *p) [inline]

This routine prints an error message to the standard error.

Parameters:

← *p* The message to print.

Definition at line 26 of file cell.hh.

The documentation for this struct was generated from the following file:

- [cell.hh](#)

2.3 neighbor_none Class Reference

A class passed to the [voronoicell_base](#) template to switch off neighbor computation.

```
#include <cell.hh>
```

Public Member Functions

- [neighbor_none](#) (voronoicell_base< [neighbor_none](#) > *ivc)
- void [allocate](#) (int i, int m)
- void [add_memory_vertices](#) (int i)
- void [add_memory_vorder](#) (int i)
- void [init](#) ()
- void [init_octahedron](#) ()
- void [init_tetrahedron](#) ()
- void [set_pointer](#) (int p, int n)
- void [copy](#) (int a, int b, int c, int d)
- void [set](#) (int a, int b, int c)
- void [set_aux1](#) (int k)
- void [copy_aux1](#) (int a, int b)

- void [copy_aux1_shift](#) (int a, int b)
- void [set_aux2_copy](#) (int a, int b)
- void [copy_pointer](#) (int a, int b)
- void [set_to_aux1](#) (int j)
- void [set_to_aux2](#) (int j)
- void [print_edges](#) (int i)
- void [allocate_aux1](#) (int i)
- void [switch_to_aux1](#) (int i)
- void [copy_to_aux1](#) (int i, int m)
- void [set_to_aux1_offset](#) (int k, int m)
- void [print](#) (ostream &os, int i, int j)
- void [label_facets](#) ()
- void [neighbors](#) (ostream &os)
- void [check_facets](#) ()

2.3.1 Detailed Description

A class passed to the [voronoicell_base](#) template to switch off neighbor computation.

This is a class full of empty routines for neighbor computation. If the [voronoicell_base](#) template is instantiated with this class, then it has the effect of switching off all neighbor computation. Since all these routines are declared inline, it should have the effect of a zero speed overhead in the resulting code.

Definition at line 252 of file cell.hh.

2.3.2 Constructor & Destructor Documentation

2.3.2.1 [neighbor_none::neighbor_none](#) ([voronoicell_base](#)< [neighbor_none](#) > * *ivc*) [inline]

This is a blank constructor.

Definition at line 255 of file cell.hh.

2.3.3 Member Function Documentation

2.3.3.1 [void neighbor_none::add_memory_vertices](#) (int *i*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 259 of file cell.hh.

2.3.3.2 [void neighbor_none::add_memory_vorder](#) (int *i*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 261 of file cell.hh.

2.3.3.3 [void neighbor_none::allocate](#) (int *i*, int *m*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 257 of file cell.hh.

2.3.3.4 void neighbor_none::allocate_aux1 (int *i*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 291 of file cell.hh.

2.3.3.5 void neighbor_none::check_facets () [inline]

This is a blank placeholder function that does nothing.

Definition at line 305 of file cell.hh.

2.3.3.6 void neighbor_none::copy (int *a*, int *b*, int *c*, int *d*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 271 of file cell.hh.

2.3.3.7 void neighbor_none::copy_aux1 (int *a*, int *b*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 277 of file cell.hh.

2.3.3.8 void neighbor_none::copy_aux1_shift (int *a*, int *b*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 279 of file cell.hh.

2.3.3.9 void neighbor_none::copy_pointer (int *a*, int *b*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 283 of file cell.hh.

2.3.3.10 void neighbor_none::copy_to_aux1 (int *i*, int *m*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 295 of file cell.hh.

2.3.3.11 void neighbor_none::init () [inline]

This is a blank placeholder function that does nothing.

Definition at line 263 of file cell.hh.

2.3.3.12 void neighbor_none::init_octahedron () [inline]

This is a blank placeholder function that does nothing.

Definition at line 265 of file cell.hh.

2.3.3.13 void neighbor_none::init_tetrahedron () [inline]

This is a blank placeholder function that does nothing.

Definition at line 267 of file cell.hh.

2.3.3.14 void neighbor_none::label_facets () [inline]

This is a blank placeholder function that does nothing.

Definition at line 301 of file cell.hh.

2.3.3.15 void neighbor_none::neighbors (ostream & os) [inline]

This is a blank placeholder function that does nothing.

Definition at line 303 of file cell.hh.

2.3.3.16 void neighbor_none::print (ostream & os, int i, int j) [inline]

This is a blank placeholder function that does nothing.

This routine is a placeholder which just prints the ID of a vertex.

Parameters:

← *&os* The output stream to write to.

← *i* The ID of a vertex.

← *j* The particular plane of interest (ignored in this routine).

Definition at line 1813 of file cell.cc.

2.3.3.17 void neighbor_none::print_edges (int i) [inline]

This is a blank placeholder function that does nothing.

Definition at line 289 of file cell.hh.

2.3.3.18 void neighbor_none::set (int a, int b, int c) [inline]

This is a blank placeholder function that does nothing.

Definition at line 273 of file cell.hh.

2.3.3.19 void neighbor_none::set_aux1 (int k) [inline]

This is a blank placeholder function that does nothing.

Definition at line 275 of file cell.hh.

2.3.3.20 void neighbor_none::set_aux2_copy (int a, int b) [inline]

This is a blank placeholder function that does nothing.

Definition at line 281 of file cell.hh.

2.3.3.21 void neighbor_none::set_pointer (int *p*, int *n*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 269 of file cell.hh.

2.3.3.22 void neighbor_none::set_to_aux1 (int *j*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 285 of file cell.hh.

2.3.3.23 void neighbor_none::set_to_aux1_offset (int *k*, int *m*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 297 of file cell.hh.

2.3.3.24 void neighbor_none::set_to_aux2 (int *j*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 287 of file cell.hh.

2.3.3.25 void neighbor_none::switch_to_aux1 (int *i*) [inline]

This is a blank placeholder function that does nothing.

Definition at line 293 of file cell.hh.

The documentation for this class was generated from the following files:

- [cell.hh](#)
- [cell.cc](#)

2.4 neighbor_track Class Reference

A class passed to the [voronoicell_base](#) template to switch on the neighbor computation.

```
#include <cell.hh>
```

Public Member Functions

- [neighbor_track](#) (voronoicell_base< [neighbor_track](#) > *ivc)
- [~neighbor_track](#) ()
- void [allocate](#) (int i, int m)
- void [add_memory_vertices](#) (int i)
- void [add_memory_vorder](#) (int i)
- void [init](#) ()
- void [init_octahedron](#) ()
- void [init_tetrahedron](#) ()
- void [set_pointer](#) (int p, int n)

- void [copy](#) (int a, int b, int c, int d)
- void [set](#) (int a, int b, int c)
- void [set_aux1](#) (int k)
- void [copy_aux1](#) (int a, int b)
- void [copy_aux1_shift](#) (int a, int b)
- void [set_aux2_copy](#) (int a, int b)
- void [copy_pointer](#) (int a, int b)
- void [set_to_aux1](#) (int j)
- void [set_to_aux2](#) (int j)
- void [print_edges](#) (int i)
- void [allocate_aux1](#) (int i)
- void [switch_to_aux1](#) (int i)
- void [copy_to_aux1](#) (int i, int m)
- void [set_to_aux1_offset](#) (int k, int m)
- void [print](#) (ostream &os, int i, int j)
- void [label_facets](#) ()
- void [neighbors](#) (ostream &os)
- void [check_facets](#) ()

Data Fields

- int ** [mne](#)
- int ** [ne](#)
- [voronoicell_base](#)< [neighbor_track](#) > * [vc](#)

2.4.1 Detailed Description

A class passed to the [voronoicell_base](#) template to switch on the neighbor computation.

This class encapsulates all the routines which are required to carry out the neighbor tracking. If the [voronoicell_base](#) template is instantiated with this class, then the neighbor computation is enabled. All these routines are simple and declared inline, so they should be directly integrated into the functions in the voronoicell class during compilation, without zero function call overhead.

Definition at line 317 of file cell.hh.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 [neighbor_track::neighbor_track](#) ([voronoicell_base](#)< [neighbor_track](#) > * *ivc*)

This constructs the [neighbor_track](#) class, within a current voronoicell_neighbor class. It allocates memory for neighbor storage in a similar way to the voronoicell constructor.

Definition at line 2022 of file cell.cc.

2.4.2.2 [neighbor_track::~~neighbor_track](#) ()

The destructor for the [neighbor_track](#) class deallocates the arrays for neighbor tracking.

Definition at line 2033 of file cell.cc.

2.4.3 Member Function Documentation

2.4.3.1 `void neighbor_track::add_memory_vertices (int i)` [inline]

This increases the size of the `ne[]` array.

Parameters:

← *i* the new size of the array.

Definition at line 2048 of file `cell.cc`.

2.4.3.2 `void neighbor_track::add_memory_vorder (int i)` [inline]

This increases the size of the maximum allowable vertex order in the neighbor tracking.

Definition at line 2057 of file `cell.cc`.

2.4.3.3 `void neighbor_track::allocate (int i, int m)` [inline]

This allocates a single array for neighbor tracking.

Parameters:

← *i* the vertex order of the array to be extended.

← *m* the size of the array to be extended.

Definition at line 2042 of file `cell.cc`.

2.4.3.4 `void neighbor_track::allocate_aux1 (int i)` [inline]

This allocates a new array and sets the auxiliary pointer to it.

Definition at line 2175 of file `cell.cc`.

2.4.3.5 `void neighbor_track::check_facets ()` [inline]

This routine checks to make sure the neighbor information of each facets is consistent.

Definition at line 2199 of file `cell.cc`.

2.4.3.6 `void neighbor_track::copy (int a, int b, int c, int d)` [inline]

This is a basic operation to copy `ne[c][d]` to `ne[a][b]`.

Definition at line 2116 of file `cell.cc`.

2.4.3.7 `void neighbor_track::copy_aux1 (int a, int b)` [inline]

This is a basic operation to copy a neighbor into `paux1`.

Definition at line 2133 of file `cell.cc`.

2.4.3.8 void neighbor_track::copy_aux1_shift (int *a*, int *b*) [inline]

This is a basic operation to copy a neighbor into paux1 with a shift. It is used in the delete_connection() routine of the voronoicell class.

Definition at line 2139 of file cell.cc.

2.4.3.9 void neighbor_track::copy_pointer (int *a*, int *b*) [inline]

This is a basic routine to copy ne[b] into ne[a].

Definition at line 2151 of file cell.cc.

2.4.3.10 void neighbor_track::copy_to_aux1 (int *i*, int *m*) [inline]

This routine copies neighbor information into the auxiliary pointer.

Definition at line 2188 of file cell.cc.

2.4.3.11 void neighbor_track::init () [inline]

This initializes the neighbor information for a rectangular box and is called during the initialization routine for the voronoicell class.

Definition at line 2067 of file cell.cc.

2.4.3.12 void neighbor_track::init_octahedron () [inline]

This initializes the neighbor information for an octahedron. The eight initial faces are assigned ID numbers from -1 to -8.

Definition at line 2084 of file cell.cc.

2.4.3.13 void neighbor_track::init_tetrahedron () [inline]

This initializes the neighbor information for a tetrahedron. The four initial faces are assigned ID numbers from -1 to -4.

Definition at line 2098 of file cell.cc.

2.4.3.14 void neighbor_track::label_facets () [inline]

This routine labels the facets in an arbitrary order, starting from one.

Definition at line 2257 of file cell.cc.

2.4.3.15 void neighbor_track::neighbors (ostream & *os*) [inline]

This routine provides a list of plane IDs.

Parameters:

← *&os* An output stream to write to.

Definition at line 2229 of file cell.cc.

2.4.3.16 `void neighbor_track::print (ostream & os, int i, int j)` [inline]

This routine prints out a bracketed pair showing a vertex number, and the corresponding neighbor information.

Parameters:

- ← *os* The output stream to write to.
- ← *i* The vertex number to print.
- ← *j* The index of the neighbor information to print.

Definition at line 2291 of file cell.cc.

2.4.3.17 `void neighbor_track::print_edges (int i)` [inline]

This prints out the neighbor information for vertex *i*.

Definition at line 2166 of file cell.cc.

2.4.3.18 `void neighbor_track::set (int a, int b, int c)` [inline]

This is a basic operation to carry out $ne[a][b]=c$.

Definition at line 2121 of file cell.cc.

2.4.3.19 `void neighbor_track::set_aux1 (int k)` [inline]

This is a basic operation to set the auxiliary pointer *paux1*.

Parameters:

- ← *k* the order of the vertex to point to.

Definition at line 2128 of file cell.cc.

2.4.3.20 `void neighbor_track::set_aux2_copy (int a, int b)` [inline]

This routine sets the second auxiliary pointer to a new section of memory, and then copies existing neighbor information into it.

Definition at line 2145 of file cell.cc.

2.4.3.21 `void neighbor_track::set_pointer (int p, int n)` [inline]

This is a basic operation to set a new pointer in the *ne[]* array.

Parameters:

- ← *p* the index in the *ne[]* array to set.
- ← *n* the order of the vertex.

Definition at line 2111 of file cell.cc.

2.4.3.22 `void neighbor_track::set_to_aux1 (int j)` [inline]

This sets `ne[j]` to the first auxiliary pointer.

Definition at line 2156 of file `cell.cc`.

2.4.3.23 `void neighbor_track::set_to_aux1_offset (int k, int m)` [inline]

This sets `ne[k]` to the auxiliary pointer with an offset.

Definition at line 2193 of file `cell.cc`.

2.4.3.24 `void neighbor_track::set_to_aux2 (int j)` [inline]

This sets `ne[j]` to the second auxiliary pointer.

Definition at line 2161 of file `cell.cc`.

2.4.3.25 `void neighbor_track::switch_to_aux1 (int i)` [inline]

This deletes a particular neighbor array and switches the pointer to the auxiliary pointer.

Definition at line 2181 of file `cell.cc`.

2.4.4 Field Documentation

2.4.4.1 `int** neighbor_track::mne`

This two dimensional array holds the neighbor information associated with each vertex. `mne[p]` is a one dimensional array which holds all of the neighbor information for vertices of order `p`.

Definition at line 323 of file `cell.hh`.

2.4.4.2 `int** neighbor_track::ne`

This is a two dimensional array that holds the neighbor information associated with each vertex. `ne[i]` points to a one-dimensional array in `mne[ne[i]]`. `ne[i][j]` holds the neighbor information associated with the `j`th edge of vertex `i`. It is set to the ID number of the plane that made the face that is clockwise from the `j`th edge.

Definition at line 330 of file `cell.hh`.

2.4.4.3 `voronoi_cell_base<neighbor_track>* neighbor_track::vc`

This is a pointer back to the `voronoi_cell` class which created this class. It is used to reference the members of that class in computations.

Definition at line 336 of file `cell.hh`.

The documentation for this class was generated from the following files:

- [cell.hh](#)
- [cell.cc](#)

2.5 radius_mono Class Reference

A class encapsulating all routines specifically needed in the standard Voronoi tessellation.

```
#include <container.hh>
```

Public Member Functions

- `radius_mono` (`container_base`< `radius_mono` > *`icc`)
- `void import` (`istream` &`is`)
- `void store_radius` (`int` `i`, `int` `j`, `fpoint` `r`)
- `void clear_max` ()
- `void init` (`int` `s`, `int` `i`)
- `fpoint volume` (`int` `ijk`, `int` `s`)
- `fpoint cutoff` (`fpoint` `lrs`)
- `fpoint scale` (`fpoint` `rs`, `int` `t`, `int` `q`)
- `void print` (`ostream` &`os`, `int` `ijk`, `int` `q`)
- `void rad` (`ostream` &`os`, `int` `l`, `int` `c`)

Data Fields

- `const int mem_size`

2.5.1 Detailed Description

A class encapsulating all routines specifically needed in the standard Voronoi tessellation.

This class encapsulates all the routines that are required for carrying out a standard Voronoi tessellation that would be appropriate for a monodisperse system. When the container class is instantiated using this class, all information about particle radii is switched off. Since all these functions are declared inline, there should be no loss of speed.

Definition at line 234 of file `container.hh`.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 `radius_mono::radius_mono (container_base< radius_mono > *icc)` [inline]

This constructor sets a pointer back to the container class that created it, and initializes the `mem_size` constant to 3.

Definition at line 243 of file `container.hh`.

2.5.3 Member Function Documentation

2.5.3.1 `void radius_mono::clear_max ()` [inline]

This is a blank placeholder function that does nothing.

Definition at line 248 of file `container.hh`.

2.5.3.2 `fpoint radius_mono::cutoff (fpoint lrs)` [inline]

This routine is called when deciding when to terminate the computation of a Voronoi cell. For the monodisperse case, this routine just returns the same value that is passed to it.

Parameters:

← *lrs* a cutoff radius for the cell computation.

Returns:

The same value passed to it.

Definition at line 1487 of file container.cc.

2.5.3.3 `void radius_mono::import (istream & is)` [inline]

Imports a list of particles from an input stream for the monodisperse case where no radius information is expected.

Parameters:

← *&is* an input stream to read from.

Definition at line 1440 of file container.cc.

2.5.3.4 `void radius_mono::init (int s, int i)` [inline]

This is a blank placeholder function that does nothing.

Definition at line 250 of file container.hh.

2.5.3.5 `void radius_mono::print (ostream & os, int ijk, int q)` [inline]

This is a blank placeholder function that does nothing.

Definition at line 255 of file container.hh.

2.5.3.6 `void radius_mono::rad (ostream & os, int l, int c)` [inline]

Prints the radius of particle, by just supplying a generic value of "s".

Parameters:

← *&os* the output stream to write to.

← *l* the region to consider.

← *c* the number of the particle within the region.

Definition at line 1495 of file container.cc.

2.5.3.7 `fpoint radius_mono::scale (fpoint rs, int t, int q)` [inline]

Applies a blank scaling to the position of a cutting plane.

Parameters:

- ← *rs* the distance between the Voronoi cell and the cutting plane.
- ← *t* the region to consider
- ← *q* the number of the particle within the region.

Returns:

The scaled position, which for this case, is equal to *rs*.

Definition at line 1542 of file container.cc.

2.5.3.8 `void radius_mono::store_radius (int i, int j, fpoint r)` [inline]

This is a blank placeholder function that does nothing.

Definition at line 246 of file container.hh.

2.5.3.9 `fpoint radius_mono::volume (int ijk, int s)` [inline]

Returns the scaled volume of a particle, which is always set to 0.125 for the monodisperse case where particles are taken to have unit diameter.

Parameters:

- ← *ijk* the region to consider.
- ← *s* the number of the particle within the region.

Returns:

The cube of the radius of the particle, which is 0.125 in this case.

Definition at line 1514 of file container.cc.

2.5.4 Field Documentation

2.5.4.1 `const int radius_mono::mem_size`

The number of floating point numbers allocated for each particle in the container, set to 3 for this case for the x, y, and z positions.

Definition at line 239 of file container.hh.

The documentation for this class was generated from the following files:

- [container.hh](#)
- [container.cc](#)

2.6 radius_poly Class Reference

A class encapsulating all routines specifically needed in the Voronoi radical tessellation.

```
#include <container.hh>
```

Public Member Functions

- `radius_poly` (`container_base`< `radius_poly` > *icc)
- `void import` (`istream` &is)
- `void store_radius` (`int` i, `int` j, `fpoint` r)
- `void clear_max` ()
- `void init` (`int` ijk, `int` s)
- `fpoint volume` (`int` ijk, `int` s)
- `fpoint cutoff` (`fpoint` lrs)
- `fpoint scale` (`fpoint` rs, `int` t, `int` q)
- `void print` (`ostream` &os, `int` ijk, `int` q)
- `void rad` (`ostream` &os, `int` l, `int` c)

Data Fields

- `const int mem_size`

2.6.1 Detailed Description

A class encapsulating all routines specifically needed in the Voronoi radical tessellation.

This class encapsulates all the routines that are required for carrying out the radical Voronoi tessellation that is appropriate for polydisperse sphere. When the container class is instantiated with this class, information about particle radii is switched on.

Definition at line 268 of file container.hh.

2.6.2 Constructor & Destructor Documentation

2.6.2.1 `radius_poly::radius_poly (container_base< radius_poly > *icc)` [inline]

This constructor sets a pointer back to the container class that created it, and initializes the `mem_size` constant to 4.

Definition at line 277 of file container.hh.

2.6.3 Member Function Documentation

2.6.3.1 `void radius_poly::clear_max ()` [inline]

Clears the stored maximum radius.

Definition at line 1433 of file container.cc.

2.6.3.2 `fpoint radius_poly::cutoff (fpoint lrs)` [inline]

This routine is called when deciding when to terminate the computation of a Voronoi cell. For the Voronoi radical tessellation for a polydisperse case, this routine multiplies the cutoff value by the scaling factor that was precomputed in the [init\(\)](#) routine.

Parameters:

← *lrs* a cutoff radius for the cell computation.

Returns:

The value scaled by the factor mul.

Definition at line 1478 of file container.cc.

2.6.3.3 `void radius_poly::import (istream & is)` [inline]

Imports a list of particles from an input stream for the polydisperse case, where both positions and particle radii are both stored.

Parameters:

← *&is* an input stream to read from.

Definition at line 1452 of file container.cc.

2.6.3.4 `void radius_poly::init (int ijk, int s)` [inline]

Initializes the [radius_poly](#) class for a new Voronoi cell calculation, by computing the radial cut-off value, based on the current particle's radius and the maximum radius of any particle in the packing.

Parameters:

← *ijk* the region to consider.

← *s* the number of the particle within the region.

Definition at line 1466 of file container.cc.

2.6.3.5 `void radius_poly::print (ostream & os, int ijk, int q)` [inline]

Prints the radius of a particle to an open file stream.

Parameters:

← *&os* an open file stream.

← *ijk* the region to consider.

← *q* the number of the particle within the region.

Definition at line 1550 of file container.cc.

2.6.3.6 void radius_poly::rad (ostream & os, int l, int c) [inline]

Prints the radius of a particle to an open output stream.

Parameters:

- ← *os* the output stream to write to.
- ← *l* the region to consider.
- ← *c* the number of the particle within the region.

Definition at line 1503 of file container.cc.

2.6.3.7 fpoint radius_poly::scale (fpoint rs, int t, int q) [inline]

Scales the position of a plane according to the relative sizes of the particle radii.

Parameters:

- ← *rs* the distance between the Voronoi cell and the cutting plane.
- ← *t* the region to consider
- ← *q* the number of the particle within the region.

Returns:

The scaled position.

Definition at line 1533 of file container.cc.

2.6.3.8 void radius_poly::store_radius (int i, int j, fpoint r) [inline]

Sets the radius of the jth particle in region i to r, and updates the maximum particle radius.

Parameters:

- ← *i* the region of the particle to consider.
- ← *j* the number of the particle within the region.
- ← *r* the radius to set.

Definition at line 1427 of file container.cc.

2.6.3.9 fpoint radius_poly::volume (int ijk, int s) [inline]

Returns the scaled volume of a particle.

Parameters:

- ← *ijk* the region to consider.
- ← *s* the number of the particle within the region.

Returns:

The cube of the radius of the particle.

Definition at line 1522 of file container.cc.

2.6.4 Field Documentation

2.6.4.1 `const int radius_poly::mem_size`

The number of floating point numbers allocated for each particle in the container, set to 4 for this case for the x, y, and z positions, plus the radius.

Definition at line 273 of file container.hh.

The documentation for this class was generated from the following files:

- [container.hh](#)
- [container.cc](#)

2.7 suretest Class Reference

A class to reliably carry out floating point comparisons, storing marginal cases for future reference.

```
#include <cell.hh>
```

Public Member Functions

- [suretest\(\)](#)
- [~suretest\(\)](#)
- void [init](#)([fpoint](#) x, [fpoint](#) y, [fpoint](#) z, [fpoint](#) rsq)
- int [test](#)(int n, [fpoint](#) &ans)

Data Fields

- [fpoint * p](#)

2.7.1 Detailed Description

A class to reliably carry out floating point comparisons, storing marginal cases for future reference.

Floating point comparisons can be unreliable on some processor architectures, and can produce unpredictable results. On a number of popular Intel processors, floating point numbers are held to higher precision when in registers than when in memory. When a register is swapped from a register to memory, a truncation error, and in some situations this can create circumstances where for two numbers c and d, the program finds $c > d$ first, but later $c < d$. The programmer has no control over when the swaps between memory and registers occur, and recompiling with slightly different code can give different results. One solution to avoid this is to force the compiler to evaluate everything in memory (e.g. by using the `-ffloat-store` option in the GNU C++ compiler) but this could be viewed overkill, since it slows the code down, and the extra register precision is useful.

In the plane cutting routine of the `voronoicell` class, we need to reliably know whether a vertex lies inside, outside, or on the cutting plane, since if it changed during the tracing process there would be confusion. This class makes these tests reliable, by storing the results of marginal cases, where the vertex lies within `tolerance2` of the cutting plane. If that vertex is tested again, then code looks up the value of the table in a buffer, rather than doing the floating point comparison again. Only vertices which are close to the plane are stored and tested, so this routine should create minimal computational overhead.

Definition at line 55 of file `cell.hh`.

2.7.2 Constructor & Destructor Documentation

2.7.2.1 `suretest::suretest ()`

Initializes the [suretest](#) class and creates a buffer for marginal points.

Definition at line 1683 of file `cell.cc`.

2.7.2.2 `suretest::~~suretest ()`

Suretest destructor to free memory allocation.

Definition at line 1688 of file `cell.cc`.

2.7.3 Member Function Documentation

2.7.3.1 `void suretest::init (fpoint x, fpoint y, fpoint z, fpoint rsq)` `[inline]`

Sets up the [suretest](#) class with a particular test plane, and removes any special cases from the table.

Definition at line 1694 of file `cell.cc`.

2.7.3.2 `int suretest::test (int n, fpoint & ans)` `[inline]`

Definition at line 1699 of file `cell.cc`.

2.7.4 Field Documentation

2.7.4.1 `fpoint* suretest::p`

This is a pointer to the array in the `voronoicell` class which holds the vertex coordinates.

Definition at line 58 of file `cell.hh`.

The documentation for this class was generated from the following files:

- [cell.hh](#)
- [cell.cc](#)

2.8 `voronoicell_base< n_option >` Class Template Reference

A class encapsulating all the routines for storing and calculating a single Voronoi cell.

```
#include <cell.hh>
```

Public Member Functions

- [voronoicell_base \(\)](#)
- [~voronoicell_base \(\)](#)
- `void init (fpoint xmax, fpoint xmin, fpoint ymax, fpoint ymin, fpoint zmin, fpoint zmax)`
- `void init_octahedron (fpoint l)`

- void `init_tetrahedron` (fpoint x0, fpoint y0, fpoint z0, fpoint x1, fpoint y1, fpoint z1, fpoint x2, fpoint y2, fpoint z2, fpoint x3, fpoint y3, fpoint z3)
- void `init_test` (int n)
- void `add_vertex` (fpoint x, fpoint y, fpoint z, int a)
- void `add_vertex` (fpoint x, fpoint y, fpoint z, int a, int b)
- void `add_vertex` (fpoint x, fpoint y, fpoint z, int a, int b, int c)
- void `add_vertex` (fpoint x, fpoint y, fpoint z, int a, int b, int c, int d)
- void `add_vertex` (fpoint x, fpoint y, fpoint z, int a, int b, int c, int d, int e)
- void `draw_pov` (ostream &os, fpoint x, fpoint y, fpoint z)
- void `draw_pov` (const char *filename, fpoint x, fpoint y, fpoint z)
- void `draw_pov` (fpoint x, fpoint y, fpoint z)
- void `draw_pov_mesh` (ostream &os, fpoint x, fpoint y, fpoint z)
- void `draw_pov_mesh` (const char *filename, fpoint x, fpoint y, fpoint z)
- void `draw_pov_mesh` (fpoint x, fpoint y, fpoint z)
- void `draw_gnuplot` (ostream &os, fpoint x, fpoint y, fpoint z)
- void `draw_gnuplot` (const char *filename, fpoint x, fpoint y, fpoint z)
- void `draw_gnuplot` (fpoint x, fpoint y, fpoint z)
- void `check_relations` ()
- void `check_duplicates` ()
- void `construct_relations` ()
- fpoint `volume` ()
- fpoint `maxradsq` ()
- int `number_of_faces` ()
- void `print_edges` ()
- void `perturb` (fpoint r)
- void `facets` (ostream &os)
- void `facets` ()
- void `facets` (const char *filename)
- void `facet_statistics` (ostream &os)
- void `facet_statistics` ()
- void `facet_statistics` (const char *filename)
- bool `nplane` (fpoint x, fpoint y, fpoint z, fpoint rs, int p_id)
- bool `nplane` (fpoint x, fpoint y, fpoint z, int p_id)
- bool `plane` (fpoint x, fpoint y, fpoint z, fpoint rs)
- bool `plane` (fpoint x, fpoint y, fpoint z)
- bool `plane_intersects` (fpoint x, fpoint y, fpoint z, fpoint rs)
- bool `plane_intersects_guess` (fpoint x, fpoint y, fpoint z, fpoint rs)
- void `label_facets` ()
- void `neighbors` (ostream &os)
- void `check_facets` ()

Data Fields

- int ** [ed](#)
- int * [nu](#)
- int [current_vertices](#)
- int [current_vertex_order](#)
- int [current_delete_size](#)
- int [current_delete2_size](#)
- [fpoint](#) * [pts](#)
- int [p](#)
- int [up](#)
- [suretest](#) [sure](#)

Friends

- class [neighbor_track](#)

2.8.1 Detailed Description

template<class [n_option](#)> class voronoicell_base< [n_option](#) >

A class encapsulating all the routines for storing and calculating a single Voronoi cell.

This class encapsulates all the routines for storing and calculating a single Voronoi cell. The cell can first be initialized by the [init\(\)](#) function to be a rectangular box. The box can then be successively cut by planes using the plane function. Other routines exist for outputting the cell, computing its volume, or finding the largest distance of a vertex from the cell center. The cell is described by two arrays. [pts\[\]](#) is a floating point array which holds the vertex positions. [ed\[\]](#) holds the table of edges, and also a relation table that determines how two vertices are connected to one another. The relation table is redundant, but helps speed up the computation. The function [check_relations\(\)](#) checks that the relational table is valid.

Definition at line 97 of file cell.hh.

2.8.2 Constructor & Destructor Documentation

2.8.2.1 template<class [n_option](#) > voronoicell_base< [n_option](#) >::voronoicell_base () [inline]

Constructs a Voronoi cell and sets up the initial memory.

Definition at line 15 of file cell.cc.

2.8.2.2 template<class [n_option](#) > voronoicell_base< [n_option](#) >::~~voronoicell_base () [inline]

The voronoicell destructor deallocates all the dynamic memory.

Definition at line 46 of file cell.cc.

2.8.3 Member Function Documentation

2.8.3.1 `template<class n_option > void voronoicell_base< n_option >::add_vertex (fpoint x, fpoint y, fpoint z, int a, int b, int c, int d, int e) [inline]`

Adds an order 5 vertex to the memory structure, and specifies its edges.

Definition at line 488 of file cell.cc.

2.8.3.2 `template<class n_option > void voronoicell_base< n_option >::add_vertex (fpoint x, fpoint y, fpoint z, int a, int b, int c, int d) [inline]`

Adds an order 4 vertex to the memory structure, and specifies its edges.

Definition at line 478 of file cell.cc.

2.8.3.3 `template<class n_option > void voronoicell_base< n_option >::add_vertex (fpoint x, fpoint y, fpoint z, int a, int b, int c) [inline]`

Adds an order 3 vertex to the memory structure, and specifies its edges.

Definition at line 468 of file cell.cc.

2.8.3.4 `template<class n_option > void voronoicell_base< n_option >::add_vertex (fpoint x, fpoint y, fpoint z, int a, int b) [inline]`

Adds an order 2 vertex to the memory structure, and specifies its edges.

Definition at line 458 of file cell.cc.

2.8.3.5 `template<class n_option > void voronoicell_base< n_option >::add_vertex (fpoint x, fpoint y, fpoint z, int a) [inline]`

Adds an order one vertex to the memory structure, and specifies its edge.

Parameters:

← (x,y,z) are the coordinates of the vertex

← a is the first and only edge of this vertex

Definition at line 448 of file cell.cc.

2.8.3.6 `template<class n_option > void voronoicell_base< n_option >::check_duplicates () [inline]`

This routine checks for any two vertices that are connected by more than one edge. The plane algorithm is designed so that this should not happen, so any occurrences are most likely errors. Note that the routine is $O(p)$, so running it every time the plane routine is called will result in a significant slowdown.

Definition at line 515 of file cell.cc.

2.8.3.7 `template<class n_option > void voronoicell_base< n_option >::check_facets () [inline]`

If the template is instantiated with the neighbor tracking turned on, then this routine will check that the neighbor information is consistent, by tracing around every facet, and ensuring that all the neighbor information for that facet refers to the same neighbor. If the neighbor tracking isn't turned on, this routine does nothing.

Definition at line 1922 of file cell.cc.

2.8.3.8 `template<class n_option > void voronoicell_base< n_option >::check_relations ()` [inline]

Checks that the relational table of the Voronoi cell is accurate, and prints out any errors. This algorithm is $O(p)$, so running it every time the plane routine is called will result in a significant slowdown.

Definition at line 500 of file cell.cc.

2.8.3.9 `template<class n_option > void voronoicell_base< n_option >::construct_relations ()` [inline]

Constructs the relational table if the edges have been specified.

Definition at line 528 of file cell.cc.

2.8.3.10 `template<class n_option > void voronoicell_base< n_option >::draw_gnuplot (fpoint x, fpoint y, fpoint z)` [inline]

An overloaded version of the draw_gnuplot routine, that prints to the standard output.

Parameters:

← (x,y,z) A displacement vector to be added to the cell's position.

Definition at line 1606 of file cell.cc.

2.8.3.11 `template<class n_option > void voronoicell_base< n_option >::draw_gnuplot (const char * filename, fpoint x, fpoint y, fpoint z)` [inline]

An overloaded version of the draw_gnuplot routine that writes directly to a file.

Parameters:

← *filename* The name of the file to write to.

← (x,y,z) A displacement vector to be added to the cell's position.

Definition at line 1594 of file cell.cc.

2.8.3.12 `template<class n_option > void voronoicell_base< n_option >::draw_gnuplot (ostream & os, fpoint x, fpoint y, fpoint z)` [inline]

Outputs the edges of the Voronoi cell (in gnuplot format) to an output stream.

Parameters:

← *&os* A reference to an output stream to write to.

← (x,y,z) A displacement vector to be added to the cell's position.

Definition at line 1577 of file cell.cc.

2.8.3.13 `template<class n_option > void voronoicell_base< n_option >::draw_pov (fpoint x , fpoint y , fpoint z)` [inline]

An overloaded version of the draw_pov routine, that outputs the edges of the Voronoi cell (in POV-Ray format) to standard output.

Parameters:

← (x,y,z) A displacement vector to be added to the cell's position.

Definition at line 1568 of file cell.cc.

2.8.3.14 `template<class n_option > void voronoicell_base< n_option >::draw_pov (const char * filename, fpoint x , fpoint y , fpoint z)` [inline]

An overloaded version of the draw_pov routine, that outputs the edges of the Voronoi cell (in POV-Ray format) to a file.

Parameters:

← *filename* The file to write to.

← (x,y,z) A displacement vector to be added to the cell's position.

Definition at line 1556 of file cell.cc.

2.8.3.15 `template<class n_option > void voronoicell_base< n_option >::draw_pov (ostream & os, fpoint x , fpoint y , fpoint z)` [inline]

Outputs the edges of the Voronoi cell (in POV-Ray format) to an open file stream, displacing the cell by an amount (x,y,z) .

Parameters:

← *&os* A output stream to write to.

← (x,y,z) A displacement vector to be added to the cell's position.

Definition at line 1538 of file cell.cc.

2.8.3.16 `template<class n_option > void voronoicell_base< n_option >::draw_pov_mesh (fpoint x , fpoint y , fpoint z)` [inline]

An overloaded version of the draw_pov_mesh routine, that prints to the standard output.

Parameters:

← (x,y,z) A displacement vector to be added to the cell's position.

Definition at line 1670 of file cell.cc.

2.8.3.17 `template<class n_option > void voronoicell_base< n_option >::draw_pov_mesh (const char * filename, fpoint x, fpoint y, fpoint z) [inline]`

An overloaded version of the `draw_pov_mesh` routine, that writes directly to a file.

Parameters:

- ← *filename* A filename to write to.
- ← *(x,y,z)* A displacement vector to be added to the cell's position.

Definition at line 1658 of file cell.cc.

2.8.3.18 `template<class n_option > void voronoicell_base< n_option >::draw_pov_mesh (ostream & os, fpoint x, fpoint y, fpoint z) [inline]`

Outputs the Voronoi cell in the POV mesh2 format, described in section 1.3.2.2 of the POV-Ray documentation. The mesh2 output consists of a list of vertex vectors, followed by a list of triangular faces. The routine also makes use of the optional `inside_vector` specification, which makes the mesh object solid, so the the POV-Ray Constructive Solid Geometry (CSG) can be applied.

Parameters:

- ← *&os* An output stream to write to.
- ← *(x,y,z)* A displacement vector to be added to the cell's position.

Definition at line 1620 of file cell.cc.

2.8.3.19 `template<class n_option > void voronoicell_base< n_option >::facet_statistics (const char * filename) [inline]`

An overloaded version of `facet_statistics()` which outputs the results to a file.

Parameters:

- ← *filename* The name of the file to write to.

Definition at line 1892 of file cell.cc.

2.8.3.20 `template<class n_option > void voronoicell_base< n_option >::facet_statistics () [inline]`

An overloaded version of `facet_statistics()` which outputs the results to standard output.

Definition at line 1884 of file cell.cc.

2.8.3.21 `template<class n_option > void voronoicell_base< n_option >::facet_statistics (ostream & os) [inline]`

Examines all the facets, and evaluates them by the number of vertices that they have.

Parameters:

- ← *&os* An open output stream to write to.

Definition at line 1838 of file cell.cc.

2.8.3.22 `template<class n_option > void voronoicell_base< n_option >::facets (const char * filename)`
[inline]

An overloaded version of [facets\(\)](#), which outputs the results to a file.

Parameters:

← *filename* The name of the file to write to.

Definition at line 1827 of file cell.cc.

2.8.3.23 `template<class n_option > void voronoicell_base< n_option >::facets ()` [inline]

An overloaded version of [facets\(\)](#) which output the results to the standard output.

Definition at line 1820 of file cell.cc.

2.8.3.24 `template<class n_option > void voronoicell_base< n_option >::facets (ostream & os)`
[inline]

Prints out a list of all the facets and their vertices. If the neighbor option is defined, it lists each cutting plane.

Definition at line 1749 of file cell.cc.

2.8.3.25 `template<class n_option > void voronoicell_base< n_option >::init (fpoint xmin, fpoint xmax, fpoint ymin, fpoint ymax, fpoint zmin, fpoint zmax)` [inline]

Initializes a Voronoi cell as a rectangular box with the given dimensions

Definition at line 198 of file cell.cc.

2.8.3.26 `template<class n_option > void voronoicell_base< n_option >::init_octahedron (fpoint l)`
[inline]

Initializes a Voronoi cell as a regular octahedron.

Parameters:

← *l* The distance from the octahedron center to a vertex. Six vertices are initialized at (-1,0,0), (1,0,0), (0,-1,0), (0,1,0), (0,0,-1), and (0,0,1).

Definition at line 228 of file cell.cc.

2.8.3.27 `template<class n_option > void voronoicell_base< n_option >::init_test (int n)` [inline]

Initializes an arbitrary test object using the [add_vertex\(\)](#) and [construct_relations\(\)](#) routines. See the source code for information about the specific objects.

Parameters:

← *n* the number of the test object (from 0 to 9)

Definition at line 279 of file cell.cc.

2.8.3.28 `template<class n_option > void voronoicell_base< n_option >::init_tetrahedron (fpoint x0, fpoint y0, fpoint z0, fpoint x1, fpoint y1, fpoint z1, fpoint x2, fpoint y2, fpoint z2, fpoint x3, fpoint y3, fpoint z3) [inline]`

Initializes a Voronoi cell as a tetrahedron. It assumes that the normal to the face for the first three vertices points inside.

Parameters:

- $(x0, y0, z0)$ A position vector for the first vertex.
- $(x1, y1, z1)$ A position vector for the second vertex.
- $(x2, y2, z2)$ A position vector for the third vertex.
- $(x3, y3, z3)$ A position vector for the fourth vertex.

Definition at line 256 of file cell.cc.

2.8.3.29 `template<class n_option > void voronoicell_base< n_option >::label_facets () [inline]`

If the template is instantiated with the neighbor tracking turned on, then this routine will label all the facets of the current cell. Otherwise this routine does nothing.

Definition at line 1903 of file cell.cc.

2.8.3.30 `template<class n_option > fpoint voronoicell_base< n_option >::maxradsq () [inline]`

Computes the maximum radius squared of a vertex from the center of the cell. It can be used to determine when enough particles have been testing an all planes that could cut the cell have been considered.

Returns:

- The maximum radius squared of a vertex.

Definition at line 1523 of file cell.cc.

2.8.3.31 `template<class n_option > void voronoicell_base< n_option >::neighbors (ostream & os) [inline]`

If the template is instantiated with the neighbor tracking turned on, then this routine will print out a list of all the neighbors of a given cell. Otherwise, this routine does nothing.

Parameters:

- $\leftarrow \&os$ An open output stream to write to.

Definition at line 1912 of file cell.cc.

2.8.3.32 `template<class n_option > bool voronoicell_base< n_option >::nplane (fpoint x, fpoint y, fpoint z, int p_id) [inline]`

This routine calculates the modulus squared of the vector before passing it to the main [nplane\(\)](#) routine with full arguments.

Parameters:

- ← (x,y,z) The vector to cut the cell by.
- ← p_id The plane ID (for neighbor tracking only).

Definition at line 1448 of file cell.cc.

2.8.3.33 `template<class n_option > bool voronoicell_base< n_option >::nplane (fpoint x, fpoint y, fpoint z, fpoint rsq, int p_id) [inline]`

Cuts the Voronoi cell by a particle whose center is at a separation of (x,y,z) from the cell center. The value of rsq should be initially set to $x^2 + y^2 + z^2$.

Definition at line 545 of file cell.cc.

2.8.3.34 `template<class n_option > int voronoicell_base< n_option >::number_of_faces () [inline]`

Returns the number of faces of a computed Voronoi cell.

Returns:

- The number of faces.

Definition at line 1781 of file cell.cc.

2.8.3.35 `template<class n_option > void voronoicell_base< n_option >::perturb (fpoint r) [inline]`

Randomly perturbs the points in the Voronoi cell by an amount r .

Definition at line 1676 of file cell.cc.

2.8.3.36 `template<class n_option > bool voronoicell_base< n_option >::plane (fpoint x, fpoint y, fpoint z) [inline]`

Cuts a Voronoi cell using the influence of a particle at (x,y,z) , first calculating the modulus squared of this vector before passing it to the main [nplane\(\)](#) routine. Zero is supplied as the plane ID, which will be ignored unless neighbor tracking is enabled.

Parameters:

- ← (x,y,z) The vector to cut the cell by.

Definition at line 1429 of file cell.cc.

2.8.3.37 `template<class n_option > bool voronoicell_base< n_option >::plane (fpoint x, fpoint y, fpoint z, fpoint rsq) [inline]`

This version of the plane routine just makes up the plane ID to be zero. It will only be referenced if neighbor tracking is enabled.

Parameters:

- ← (x,y,z) The vector to cut the cell by.
- ← rsq The modulus squared of the vector.

Definition at line 1439 of file cell.cc.

2.8.3.38 `template<class n_option > bool voronoicell_base< n_option >::plane_intersects (fpoint x, fpoint y, fpoint z, fpoint rsq) [inline]`

This routine tests to see whether the cell intersects a plane by starting from the guess point up. If up intersects, then it immediately returns true. Otherwise, it calls the `plane_intersects_track()` routine.

Parameters:

- ← (x,y,z) The normal vector to the plane.
- ← rsq The distance along this vector of the plane.

Returns:

false if the plane does not intersect the plane, true if it does.

Definition at line 1933 of file cell.cc.

2.8.3.39 `template<class n_option > bool voronoicell_base< n_option >::plane_intersects_guess (fpoint x, fpoint y, fpoint z, fpoint rsq) [inline]`

This routine tests to see if a cell intersects a plane. It first tests a random sample of approximately $\sqrt{p}/4$ points. If any of those are intersect, then it immediately returns true. Otherwise, it takes the closest point and passes that to `plane_intersect_track()` routine.

Parameters:

- ← (x,y,z) The normal vector to the plane.
- ← rsq The distance along this vector of the plane.

Returns:

false if the plane does not intersect the plane, true if it does.

Definition at line 1947 of file cell.cc.

2.8.3.40 `template<class n_option > void voronoicell_base< n_option >::print_edges () [inline]`

Prints the vertices, their edges, the relation table, and also notifies if any glaring memory errors are visible.

Definition at line 1730 of file cell.cc.

2.8.3.41 `template<class n_option > fpoint voronoicell_base< n_option >::volume () [inline]`

Calculates the volume of the Voronoi cell, by decomposing the cell into tetrahedra extending outward from the zeroth vertex, which are evaluated using a scalar triple product.

Returns:

A floating point number holding the calculated volume.

Definition at line 1478 of file cell.cc.

2.8.4 Field Documentation**2.8.4.1 `template<class n_option > int voronoicell_base< n_option >::current_delete2_size`**

This sets the size of the auxiliary delete stack.

Definition at line 135 of file cell.hh.

2.8.4.2 `template<class n_option > int voronoicell_base< n_option >::current_delete_size`

This sets the size of the main delete stack.

Definition at line 133 of file cell.hh.

2.8.4.3 `template<class n_option > int voronoicell_base< n_option >::current_vertex_order`

This holds the current maximum allowed order of a vertex, which sets the size of the mem, mep, and mec arrays. If a vertex is created with more vertices than this, the arrays are dynamically extended using the `add_memory_vorder` routine.

Definition at line 131 of file cell.hh.

2.8.4.4 `template<class n_option > int voronoicell_base< n_option >::current_vertices`

This holds the current size of the arrays `ed` and `nu`, which hold the vertex information. If more vertices are created than can fit in this array, then it is dynamically extended using the `add_memory_vertices` routine.

Definition at line 125 of file cell.hh.

2.8.4.5 `template<class n_option > int voronoicell_base< n_option >::ed`**

This is a two dimensional array that holds information about the edge connections of the vertices that make up the cell. The two dimensional array is not allocated in the usual method. To account for the fact the different vertices have different orders, and thus require different amounts of storage, the elements of `ed[i]` point to one-dimensional arrays in the `mep[]` array of different sizes.

More specifically, if vertex `i` has order `m`, then `ed[i]` points to a one-dimensional array in `mep[m]` that has `2*m+1` entries. The first `m` elements hold the neighboring edges, so that the `j`th edge of vertex `i` is held in `ed[i][j]`. The next `m` elements hold a table of relations which is redundant but helps speed up the computation. It satisfies the relation `ed[ed[i][j]][ed[i][m+j]]=i`. The final entry holds a back pointer, so that `ed[i+2*m]=i`. These are used when rearranging the memory.

Definition at line 116 of file cell.hh.

2.8.4.6 `template<class n_option > int* voronoicell_base< n_option >::nu`

This array holds the order of the vertices in the Voronoi cell. This array is dynamically allocated, with its current size held by `current_vertices`.

Definition at line 120 of file cell.hh.

2.8.4.7 `template<class n_option > int voronoicell_base< n_option >::p`

This sets the total number of vertices in the current cell.

Definition at line 141 of file cell.hh.

2.8.4.8 `template<class n_option > fpoint* voronoicell_base< n_option >::pts`

This is an array with size `3*current_vertices` for holding the positions of the vertices.

Definition at line 138 of file cell.hh.

2.8.4.9 `template<class n_option > suretest voronoicell_base< n_option >::sure`

This is a class used in the plane routine for carrying out reliable comparisons of whether points in the cell are inside, outside, or on the current cutting plane.

Definition at line 152 of file cell.hh.

2.8.4.10 `template<class n_option > int voronoicell_base< n_option >::up`

This is the index of particular point in the cell, which is used to start the tracing routines for plane intersection and cutting. These routines will work starting from any point, but it's often most efficient to start from the last point considered, since in many cases, the cell construction algorithm may consider many planes with similar vectors concurrently.

Definition at line 148 of file cell.hh.

The documentation for this class was generated from the following files:

- [cell.hh](#)
- [cell.cc](#)

2.9 voropp_loop Class Reference

A class to handle loops on regions of the container handling non-periodic and periodic boundary conditions.

```
#include <container.hh>
```

Public Member Functions

- `template<class r_option > voropp_loop (container_base< r_option > *q)`
- `int init (fpoint vx, fpoint vy, fpoint vz, fpoint r, fpoint &px, fpoint &py, fpoint &pz)`
- `int init (fpoint xmin, fpoint xmax, fpoint ymin, fpoint ymax, fpoint zmin, fpoint zmax, fpoint &px, fpoint &py, fpoint &pz)`
- `int inc (fpoint &px, fpoint &py, fpoint &pz)`

Data Fields

- int [ip](#)
- int [jp](#)
- int [kp](#)

2.9.1 Detailed Description

A class to handle loops on regions of the container handling non-periodic and periodic boundary conditions.

Many of the container routines require scanning over a rectangular sub-grid of blocks, and the routines for handling this are stored in the [voropp_loop](#) class. A [voropp_loop](#) class can first be initialized to either calculate the subgrid which is within a distance r of a vector (v_x, v_y, v_z) , or a subgrid corresponding to a rectangular box. The routine [inc\(\)](#) can then be successively called to step through all the blocks within this subgrid.

Definition at line 302 of file container.hh.

2.9.2 Constructor & Destructor Documentation

2.9.2.1 `template<class r_option > voropp_loop::voropp_loop (container_base< r_option > * q)` [\[inline\]](#)

Creates a [voropp_loop](#) object, by pulling the necessary constants about the container geometry from a pointer to the current container class.

Definition at line 1300 of file container.cc.

2.9.3 Member Function Documentation

2.9.3.1 `int voropp_loop::inc (fpoint & px, fpoint & py, fpoint & pz)` [\[inline\]](#)

Returns the next block to be tested in a loop, and updates the periodicity vector if necessary.

Definition at line 1375 of file container.cc.

2.9.3.2 `int voropp_loop::init (fpoint xmin, fpoint xmax, fpoint ymin, fpoint ymax, fpoint zmin, fpoint zmax, fpoint & px, fpoint & py, fpoint & pz)` [\[inline\]](#)

Initializes a [voropp_loop](#) object, by finding all blocks which overlap the box with corners $(x_{min}, y_{min}, z_{min})$ and $(x_{max}, y_{max}, z_{max})$. It returns the first block which is to be tested, and sets the periodic displacement vector (p_x, p_y, p_z) accordingly.

Definition at line 1343 of file container.cc.

2.9.3.3 `int voropp_loop::init (fpoint vx, fpoint vy, fpoint vz, fpoint r, fpoint & px, fpoint & py, fpoint & pz)` [\[inline\]](#)

Initializes a [voropp_loop](#) object, by finding all blocks which are within a distance r of the vector (v_x, v_y, v_z) . It returns the first block which is to be tested, and sets the periodic displacement vector (p_x, p_y, p_z) accordingly.

Definition at line 1309 of file container.cc.

2.9.4 Field Documentation

2.9.4.1 `int voropp_loop::ip`

The current block index in the x direction, referencing a real cell in the range 0 to nx-1.

Definition at line 311 of file container.hh.

2.9.4.2 `int voropp_loop::jp`

The current block index in the y direction, referencing a real cell in the range 0 to ny-1.

Definition at line 314 of file container.hh.

2.9.4.3 `int voropp_loop::kp`

The current block index in the z direction, referencing a real cell in the range 0 to nz-1.

Definition at line 317 of file container.hh.

The documentation for this class was generated from the following files:

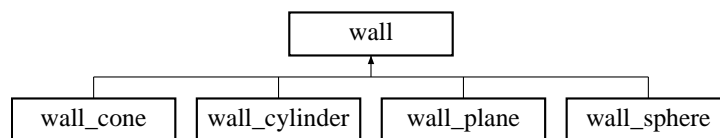
- [container.hh](#)
- [container.cc](#)

2.10 wall Class Reference

Pure virtual class from which [wall](#) objects are derived.

```
#include <container.hh>
```

Inheritance diagram for wall::



Public Member Functions

- virtual bool [point_inside](#) ([fpoint](#) x, [fpoint](#) y, [fpoint](#) z)=0
- virtual bool [cut_cell](#) ([voronoicell_base](#)< [neighbor_none](#) > &c, [fpoint](#) x, [fpoint](#) y, [fpoint](#) z)=0
- virtual bool [cut_cell](#) ([voronoicell_base](#)< [neighbor_track](#) > &c, [fpoint](#) x, [fpoint](#) y, [fpoint](#) z)=0

2.10.1 Detailed Description

Pure virtual class from which [wall](#) objects are derived.

This is a pure virtual class for a generic [wall](#) object. A [wall](#) object can be specified by deriving a new class from this and specifying the functions.

Definition at line 335 of file container.hh.

2.10.2 Member Function Documentation

2.10.2.1 `virtual bool wall::cut_cell (voronoicell_base< neighbor_track > &c, fpoint x, fpoint y, fpoint z)` [pure virtual]

A pure virtual function for cutting a cell with neighbor-tracking enabled with a [wall](#).

Implemented in [wall_sphere](#), [wall_plane](#), [wall_cylinder](#), and [wall_cone](#).

2.10.2.2 `virtual bool wall::cut_cell (voronoicell_base< neighbor_none > &c, fpoint x, fpoint y, fpoint z)` [pure virtual]

A pure virtual function for cutting a cell without neighbor-tracking with a [wall](#).

Implemented in [wall_sphere](#), [wall_plane](#), [wall_cylinder](#), and [wall_cone](#).

2.10.2.3 `virtual bool wall::point_inside (fpoint x, fpoint y, fpoint z)` [pure virtual]

A pure virtual function for testing whether a point is inside the [wall](#) object.

Implemented in [wall_sphere](#), [wall_plane](#), [wall_cylinder](#), and [wall_cone](#).

The documentation for this class was generated from the following file:

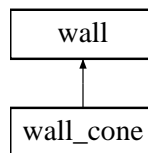
- [container.hh](#)

2.11 wall_cone Struct Reference

A class representing a conical [wall](#) object.

```
#include <wall.hh>
```

Inheritance diagram for wall_cone::



Public Member Functions

- [wall_cone](#) ([fpoint](#) ixc, [fpoint](#) iyc, [fpoint](#) izc, [fpoint](#) ixa, [fpoint](#) iya, [fpoint](#) iza, [fpoint](#) ang, int iw_id=-99)
- bool [point_inside](#) ([fpoint](#) x, [fpoint](#) y, [fpoint](#) z)
- template<class n_option >
 bool [cut_cell_base](#) (voronoicell_base< n_option > &c, [fpoint](#) x, [fpoint](#) y, [fpoint](#) z)
- bool [cut_cell](#) (voronoicell_base< neighbor_none > &c, [fpoint](#) x, [fpoint](#) y, [fpoint](#) z)
- bool [cut_cell](#) (voronoicell_base< neighbor_track > &c, [fpoint](#) x, [fpoint](#) y, [fpoint](#) z)

2.11.1 Detailed Description

A class representing a conical [wall](#) object.

This class represents a cone [wall](#) object.

Definition at line 88 of file wall.hh.

2.11.2 Constructor & Destructor Documentation

2.11.2.1 `wall_cone::wall_cone (fpoint ixc, fpoint iyc, fpoint izc, fpoint ixa, fpoint iya, fpoint iza, fpoint ang, int iw_id = -99) [inline]`

Constructs a cone [wall](#) object.

Parameters:

- ← *(ixc,iyc,izc)* the apex of the cone.
- ← *(ixa,iya,iza)* a vector pointing along the axis of the cone.
- ← *ang* the angle (in radians) of the cone, measured from the axis.
- ← *iw_id* an ID number to associate with the [wall](#) for neighbor tracking.

Definition at line 98 of file wall.hh.

2.11.3 Member Function Documentation

2.11.3.1 `bool wall_cone::cut_cell (voronoicell_base< neighbor_track > & c, fpoint x, fpoint y, fpoint z) [inline, virtual]`

A pure virtual function for cutting a cell with neighbor-tracking enabled with a [wall](#).

Implements [wall](#).

Definition at line 106 of file wall.hh.

2.11.3.2 `bool wall_cone::cut_cell (voronoicell_base< neighbor_none > & c, fpoint x, fpoint y, fpoint z) [inline, virtual]`

A pure virtual function for cutting a cell without neighbor-tracking with a [wall](#).

Implements [wall](#).

Definition at line 105 of file wall.hh.

2.11.3.3 `template<class n_option > bool wall_cone::cut_cell_base (voronoicell_base< n_option > & c, fpoint x, fpoint y, fpoint z) [inline]`

Cuts a cell by the cone [wall](#) object. The conical [wall](#) is approximated by a single plane applied at the point on the cone which is closest to the center of the cell. This works well for particle arrangements that are packed against the [wall](#), but loses accuracy for sparse particle distributions.

Parameters:

- ← *&c* the Voronoi cell to be cut.

$\leftarrow (x,y,z)$ the location of the Voronoi cell.

Returns:

true if the cell still exists, false if the cell is deleted.

Definition at line 108 of file wall.cc.

2.11.3.4 bool wall_cone::point_inside (fpoint x, fpoint y, fpoint z) [virtual]

Tests to see whether a point is inside the cone [wall](#) object.

Parameters:

$\leftarrow (x,y,z)$ the vector to test.

Returns:

true if the point is inside, false if the point is outside.

Implements [wall](#).

Definition at line 88 of file wall.cc.

The documentation for this struct was generated from the following files:

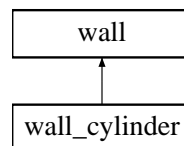
- [wall.hh](#)
- [wall.cc](#)

2.12 wall_cylinder Struct Reference

A class representing a cylindrical [wall](#) object.

```
#include <wall.hh>
```

Inheritance diagram for wall_cylinder::



Public Member Functions

- [wall_cylinder](#) (fpoint ixc, fpoint iyc, fpoint izc, fpoint ixa, fpoint iya, fpoint iza, fpoint irc, int iw_id=-99)
- bool [point_inside](#) (fpoint x, fpoint y, fpoint z)
- template<class n_option >
bool [cut_cell_base](#) (voronoi_cell_base< n_option > &c, fpoint x, fpoint y, fpoint z)
- bool [cut_cell](#) (voronoi_cell_base< neighbor_none > &c, fpoint x, fpoint y, fpoint z)
- bool [cut_cell](#) (voronoi_cell_base< neighbor_track > &c, fpoint x, fpoint y, fpoint z)

2.12.1 Detailed Description

A class representing a cylindrical [wall](#) object.

This class represents a open cylinder [wall](#) object.

Definition at line 61 of file wall.hh.

2.12.2 Constructor & Destructor Documentation

2.12.2.1 `wall_cylinder::wall_cylinder (fpoint ixc, fpoint iyc, fpoint izc, fpoint ixa, fpoint iya, fpoint iza, fpoint irc, int iw_id = -99) [inline]`

Constructs a cylinder [wall](#) object.

Parameters:

- ← *(ixc,iyc,izc)* a point on the axis of the cylinder.
- ← *(ixa,iya,iza)* a vector pointing along the direction of the cylinder.
- ← *irc* the radius of the cylinder
- ← *iw_id* an ID number to associate with the [wall](#) for neighbor tracking.

Definition at line 71 of file wall.hh.

2.12.3 Member Function Documentation

2.12.3.1 `bool wall_cylinder::cut_cell (voronoicell_base< neighbor_track > & c, fpoint x, fpoint y, fpoint z) [inline, virtual]`

A pure virtual function for cutting a cell with neighbor-tracking enabled with a [wall](#).

Implements [wall](#).

Definition at line 78 of file wall.hh.

2.12.3.2 `bool wall_cylinder::cut_cell (voronoicell_base< neighbor_none > & c, fpoint x, fpoint y, fpoint z) [inline, virtual]`

A pure virtual function for cutting a cell without neighbor-tracking with a [wall](#).

Implements [wall](#).

Definition at line 77 of file wall.hh.

2.12.3.3 `template<class n_option > bool wall_cylinder::cut_cell_base (voronoicell_base< n_option > & c, fpoint x, fpoint y, fpoint z) [inline]`

Cuts a cell by the cylindrical [wall](#) object. The cylindrical [wall](#) is approximated by a single plane applied at the point on the cylinder which is closest to the center of the cell. This works well for particle arrangements that are packed against the [wall](#), but loses accuracy for sparse particle distributions.

Parameters:

- ← *&c* the Voronoi cell to be cut.

← (x,y,z) the location of the Voronoi cell.

Returns:

true if the cell still exists, false if the cell is deleted.

Definition at line 73 of file wall.cc.

2.12.3.4 bool wall_cylinder::point_inside (fpoint x, fpoint y, fpoint z) [virtual]

Tests to see whether a point is inside the cylindrical [wall](#) object.

Parameters:

← (x,y,z) the vector to test.

Returns:

true if the point is inside, false if the point is outside.

Implements [wall](#).

Definition at line 57 of file wall.cc.

The documentation for this struct was generated from the following files:

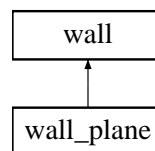
- [wall.hh](#)
- [wall.cc](#)

2.13 wall_plane Struct Reference

A class representing a plane [wall](#) object.

```
#include <wall.hh>
```

Inheritance diagram for wall_plane::



Public Member Functions

- [wall_plane](#) (fpoint ixc, fpoint iyc, fpoint izc, fpoint iac, int iw_id=-99)
- bool [point_inside](#) (fpoint x, fpoint y, fpoint z)
- template<class n_option >
bool [cut_cell_base](#) (voronoi_cell_base< n_option > &c, fpoint x, fpoint y, fpoint z)
- bool [cut_cell](#) (voronoi_cell_base< neighbor_none > &c, fpoint x, fpoint y, fpoint z)
- bool [cut_cell](#) (voronoi_cell_base< neighbor_track > &c, fpoint x, fpoint y, fpoint z)

2.13.1 Detailed Description

A class representing a plane [wall](#) object.

This class represents a single plane [wall](#) object.

Definition at line 39 of file wall.hh.

2.13.2 Constructor & Destructor Documentation

2.13.2.1 `wall_plane::wall_plane (fpoint ixc, fpoint iyc, fpoint izc, fpoint iac, int iw_id = -99)`
[inline]

Constructs a plane [wall](#) object

Parameters:

- ← (*ixc*,*iy**c*,*izc*) a normal vector to the plane.
- ← *i**ac* a displacement along the normal vector.
- ← *i**w_id* an ID number to associate with the [wall](#) for neighbor tracking.

Definition at line 46 of file wall.hh.

2.13.3 Member Function Documentation

2.13.3.1 `bool wall_plane::cut_cell (voronoicell_base< neighbor_track > & c, fpoint x, fpoint y, fpoint z)` [inline, virtual]

A pure virtual function for cutting a cell with neighbor-tracking enabled with a [wall](#).

Implements [wall](#).

Definition at line 52 of file wall.hh.

2.13.3.2 `bool wall_plane::cut_cell (voronoicell_base< neighbor_none > & c, fpoint x, fpoint y, fpoint z)` [inline, virtual]

A pure virtual function for cutting a cell without neighbor-tracking with a [wall](#).

Implements [wall](#).

Definition at line 51 of file wall.hh.

2.13.3.3 `template<class n_option> bool wall_plane::cut_cell_base (voronoicell_base< n_option > & c, fpoint x, fpoint y, fpoint z)` [inline]

Cuts a cell by the plane [wall](#) object.

Parameters:

- ← &*c* the Voronoi cell to be cut.
- ← (*x*,*y*,*z*) the location of the Voronoi cell.

Returns:

true if the cell still exists, false if the cell is deleted.

Definition at line 49 of file wall.cc.

2.13.3.4 bool wall_plane::point_inside (fpoint x, fpoint y, fpoint z) [virtual]

Tests to see whether a point is inside the plane [wall](#) object.

Parameters:

$\leftarrow (x,y,z)$ the vector to test.

Returns:

true if the point is inside, false if the point is outside.

Implements [wall](#).

Definition at line 40 of file wall.cc.

The documentation for this struct was generated from the following files:

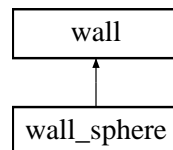
- [wall.hh](#)
- [wall.cc](#)

2.14 wall_sphere Struct Reference

A class representing a spherical [wall](#) object.

```
#include <wall.hh>
```

Inheritance diagram for wall_sphere::

**Public Member Functions**

- [wall_sphere](#) ([fpoint](#) ixc, [fpoint](#) iyc, [fpoint](#) izc, [fpoint](#) irc, int iw_id=-99)
- bool [point_inside](#) ([fpoint](#) x, [fpoint](#) y, [fpoint](#) z)
- template<class n_option >
bool [cut_cell_base](#) ([voronoicell_base](#)< n_option > &c, [fpoint](#) x, [fpoint](#) y, [fpoint](#) z)
- bool [cut_cell](#) ([voronoicell_base](#)< [neighbor_none](#) > &c, [fpoint](#) x, [fpoint](#) y, [fpoint](#) z)
- bool [cut_cell](#) ([voronoicell_base](#)< [neighbor_track](#) > &c, [fpoint](#) x, [fpoint](#) y, [fpoint](#) z)

2.14.1 Detailed Description

A class representing a spherical [wall](#) object.

This class represents a spherical [wall](#) object.

Definition at line 16 of file wall.hh.

2.14.2 Constructor & Destructor Documentation

2.14.2.1 `wall_sphere::wall_sphere (fpoint ixc, fpoint iyc, fpoint izc, fpoint irc, int iw_id = -99)`
[inline]

Constructs a spherical [wall](#) object.

Parameters:

- ← *iw_id* an ID number to associate with the [wall](#) for neighbor tracking.
- ← (*ixc, iyc, izc*) a position vector for the sphere's center.
- ← *irc* the radius of the sphere.

Definition at line 24 of file wall.hh.

2.14.3 Member Function Documentation

2.14.3.1 `bool wall_sphere::cut_cell (voronoicell_base< neighbor_track > &c, fpoint x, fpoint y, fpoint z)` [inline, virtual]

A pure virtual function for cutting a cell with neighbor-tracking enabled with a [wall](#).

Implements [wall](#).

Definition at line 30 of file wall.hh.

2.14.3.2 `bool wall_sphere::cut_cell (voronoicell_base< neighbor_none > &c, fpoint x, fpoint y, fpoint z)` [inline, virtual]

A pure virtual function for cutting a cell without neighbor-tracking with a [wall](#).

Implements [wall](#).

Definition at line 29 of file wall.hh.

2.14.3.3 `template<class n_option > bool wall_sphere::cut_cell_base (voronoicell_base< n_option > &c, fpoint x, fpoint y, fpoint z)` [inline]

Cuts a cell by the sphere [wall](#) object. The spherical [wall](#) is approximated by a single plane applied at the point on the sphere which is closest to the center of the cell. This works well for particle arrangements that are packed against the [wall](#), but loses accuracy for sparse particle distributions.

Parameters:

- ← &c the Voronoi cell to be cut.
- ← (*x, y, z*) the location of the Voronoi cell.

Returns:

true if the cell still exists, false if the cell is deleted.

Definition at line 27 of file wall.cc.

2.14.3.4 bool wall_sphere::point_inside (fpoint x, fpoint y, fpoint z) [virtual]

Tests to see whether a point is inside the sphere [wall](#) object.

Parameters:

$\leftarrow (x,y,z)$ the vector to test.

Returns:

true if the point is inside, false if the point is outside.

Implements [wall](#).

Definition at line 15 of file wall.cc.

The documentation for this struct was generated from the following files:

- [wall.hh](#)
- [wall.cc](#)

3 File Documentation

3.1 cell.cc File Reference

Function implementations for the [voronoicell_base](#) template and related classes.

```
#include "cell.hh"
```

3.1.1 Detailed Description

Function implementations for the [voronoicell_base](#) template and related classes.

Definition in file [cell.cc](#).

3.2 cell.hh File Reference

Header file for the [voronoicell_base](#) template and related classes.

```
#include "config.hh"
#include <cstdio>
#include <iostream>
#include <fstream>
#include <cmath>
```

Data Structures

- struct [fatal_error](#)
Structure for printing fatal error messages and exiting.
- class [suretest](#)
A class to reliably carry out floating point comparisons, storing marginal cases for future reference.
- class [voronoicell_base< n_option >](#)
A class encapsulating all the routines for storing and calculating a single Voronoi cell.
- class [neighbor_none](#)
A class passed to the [voronoicell_base](#) template to switch off neighbor computation.
- class [neighbor_track](#)
A class passed to the [voronoicell_base](#) template to switch on the neighbor computation.

Typedefs

- typedef [voronoicell_base< neighbor_none > voronoicell](#)
- typedef [voronoicell_base< neighbor_track > voronoicell_neighbor](#)

3.2.1 Detailed Description

Header file for the [voronoicell_base](#) template and related classes.

Definition in file [cell.hh](#).

3.2.2 Typedef Documentation

3.2.2.1 typedef [voronoicell_base<neighbor_none> voronoicell](#)

The basic voronoicell class.

Definition at line 372 of file [cell.hh](#).

3.2.2.2 typedef [voronoicell_base<neighbor_track> voronoicell_neighbor](#)

A neighbor-tracking version of the voronoicell.

Definition at line 375 of file [cell.hh](#).

3.3 [config.hh](#) File Reference

Master configuration file for setting various compile-time options.

Defines

- `#define VOROPP_VERBOSE 0`

Typedefs

- `typedef double fpoint`

Variables

- `const int init_vertices = 256`
- `const int init_vertex_order = 64`
- `const int init_3_vertices = 256`
- `const int init_n_vertices = 8`
- `const int init_marginal = 256`
- `const int init_delete_size = 256`
- `const int init_delete2_size = 256`
- `const int init_facet_size = 32`
- `const int init_wall_size = 32`
- `const int max_vertices = 16777216`
- `const int max_vertex_order = 2048`
- `const int max_n_vertices = 16777216`
- `const int max_marginal = 16777216`
- `const int max_delete_size = 16777216`
- `const int max_delete2_size = 16777216`
- `const int max_particle_memory = 16777216`
- `const int max_wall_size = 2048`
- `const fpoint tolerance = 1e-10`
- `const fpoint tolerance2 = 2e-10`
- `const fpoint large_number = 1e30`

3.3.1 Detailed Description

Master configuration file for setting various compile-time options.

Definition in file [config.hh](#).

3.3.2 Define Documentation

3.3.2.1 `#define VOROPP_VERBOSE 0`

Voro++ can print a number of different status and debugging messages to notify the user of special behavior, and this macro sets the amount which are displayed. At level 0, no messages are printed. At level 1, messages about unusual cases during cell construction are printed, such as when the plane routine bails out due to floating point problems. At level 2, general messages about memory expansion are printed. At level 3, technical details about memory management are printed.

Definition at line 62 of file [config.hh](#).

3.3.3 Typedef Documentation

3.3.3.1 typedef double fpoint

The declaration of fpoint allows that code to be compiled both using single precision numbers and double precision numbers. Under normal usage fpoint is set be a double precision floating point number, but defining the preprocessor macro VOROPP_SINGLE_PRECISION will switch it to single precision and make the code tolerances larger.

Definition at line 73 of file config.hh.

3.3.4 Variable Documentation

3.3.4.1 const int init_3_vertices = 256

The initial memory allocation for the number of regular vertices of order 3.

Definition at line 20 of file config.hh.

3.3.4.2 const int init_delete2_size = 256

The initial size for the auxiliary delete stack.

Definition at line 29 of file config.hh.

3.3.4.3 const int init_delete_size = 256

The initial size for the delete stack.

Definition at line 27 of file config.hh.

3.3.4.4 const int init_facet_size = 32

The initial size for the facets evaluation.

Definition at line 31 of file config.hh.

3.3.4.5 const int init_marginal = 256

The initial buffer size for marginal cases used by the [suretest](#) class.

Definition at line 25 of file config.hh.

3.3.4.6 const int init_n_vertices = 8

The initial memory allocation for the number of vertices of higher order.

Definition at line 23 of file config.hh.

3.3.4.7 const int init_vertex_order = 64

The initial memory allocation for the maximum vertex order.

Definition at line 17 of file config.hh.

3.3.4.8 `const int init_vertices = 256`

The initial memory allocation for the number of vertices.

Definition at line 15 of file config.hh.

3.3.4.9 `const int init_wall_size = 32`

The initial size for the [wall](#) pointer array.

Definition at line 33 of file config.hh.

3.3.4.10 `const fpoint large_number = 1e30`

A large number that is used in the computation.

Definition at line 95 of file config.hh.

3.3.4.11 `const int max_delete2_size = 16777216`

The maximum size for the auxiliary delete stack.

Definition at line 48 of file config.hh.

3.3.4.12 `const int max_delete_size = 16777216`

The maximum size for the delete stack.

Definition at line 46 of file config.hh.

3.3.4.13 `const int max_marginal = 16777216`

The maximum buffer size for marginal cases used by the [suretest](#) class.

Definition at line 44 of file config.hh.

3.3.4.14 `const int max_n_vertices = 16777216`

The maximum memory allocation for the any particular order of vertex.

Definition at line 42 of file config.hh.

3.3.4.15 `const int max_particle_memory = 16777216`

The maximum amount of particle memory allocated for a single region.

Definition at line 50 of file config.hh.

3.3.4.16 `const int max_vertex_order = 2048`

The maximum memory allocation for the maximum vertex order.

Definition at line 40 of file config.hh.

3.3.4.17 `const int max_vertices = 16777216`

The maximum memory allocation for the number of vertices.

Definition at line 38 of file `config.hh`.

3.3.4.18 `const int max_wall_size = 2048`

The maximum size for the `wall` pointer array.

Definition at line 52 of file `config.hh`.

3.3.4.19 `const fpoint tolerance = 1e-10`

If a point is within this distance of a cutting plane, then the code assumes that point exactly lies on the plane.

Definition at line 81 of file `config.hh`.

3.3.4.20 `const fpoint tolerance2 = 2e-10`

If a point is within this distance of a cutting plane, then the code stores whether this point is inside, outside, or exactly on the cutting plane in the marginal cases buffer, to prevent the test giving a different result on a subsequent evaluation due to floating point rounding errors.

Definition at line 91 of file `config.hh`.

3.4 `container.cc` File Reference

Function implementations for the `container_base` template and related classes.

```
#include "cell.hh"
#include "container.hh"
#include "worklist.cc"
```

3.4.1 Detailed Description

Function implementations for the `container_base` template and related classes.

Definition in file `container.cc`.

3.5 `container.hh` File Reference

Header file for the `container_base` template and related classes.

```
#include "config.hh"
#include <cstdio>
#include <iostream>
#include <fstream>
#include <cmath>
```

```
#include "worklist.hh"
```

Data Structures

- class [container_base< r_option >](#)
A class representing the whole simulation region.
- class [radius_mono](#)
A class encapsulating all routines specifically needed in the standard Voronoi tessellation.
- class [radius_poly](#)
A class encapsulating all routines specifically needed in the Voronoi radical tessellation.
- class [voropp_loop](#)
A class to handle loops on regions of the container handling non-periodic and periodic boundary conditions.
- class [wall](#)
Pure virtual class from which [wall](#) objects are derived.

Typedefs

- typedef [container_base< radius_mono >](#) [container](#)
- typedef [container_base< radius_poly >](#) [container_poly](#)

3.5.1 Detailed Description

Header file for the [container_base](#) template and related classes.

Definition in file [container.hh](#).

3.5.2 Typedef Documentation

3.5.2.1 typedef [container_base<radius_mono>](#) [container](#)

The basic container class.

Definition at line 350 of file [container.hh](#).

3.5.2.2 typedef [container_base<radius_poly>](#) [container_poly](#)

The polydisperse container class.

Definition at line 353 of file [container.hh](#).

3.6 voro++.cc File Reference

A file that loads all of the function implementation files.

```
#include "cell.cc"
#include "container.cc"
#include "wall.cc"
```

3.6.1 Detailed Description

A file that loads all of the function implementation files.

Definition in file [voro++.cc](#).

3.7 voro++.hh File Reference

A file that loads all of the Voropp header files.

```
#include "cell.hh"
#include "container.hh"
#include "wall.hh"
```

3.7.1 Detailed Description

A file that loads all of the Voropp header files.

Definition in file [voro++.hh](#).

3.8 wall.cc File Reference

Function implementations for the derived [wall](#) classes.

```
#include "wall.hh"
```

3.8.1 Detailed Description

Function implementations for the derived [wall](#) classes.

Definition in file [wall.cc](#).

3.9 wall.hh File Reference

Header file for the derived [wall](#) classes.

Data Structures

- struct [wall_sphere](#)

A class representing a spherical [wall](#) object.

- struct [wall_plane](#)

A class representing a plane [wall](#) object.

- struct [wall_cylinder](#)

A class representing a cylindrical [wall](#) object.

- struct [wall_cone](#)

A class representing a conical [wall](#) object.

3.9.1 Detailed Description

Header file for the derived [wall](#) classes.

Definition in file [wall.hh](#).

3.10 worklist.cc File Reference

The table of block worklists that are used during the cell computation.

3.10.1 Detailed Description

The table of block worklists that are used during the cell computation.

This file is automatically generated by `worklist_generate.pl` and it is not intended to be edited by hand.

Definition in file [worklist.cc](#).

3.11 worklist.hh File Reference

Header file for setting constants used in the block worklists that are used during cell computation.

3.11.1 Detailed Description

Header file for setting constants used in the block worklists that are used during cell computation.

This file is automatically generated by `worklist_generate.pl` and it is not intended to be edited by hand.

Definition in file [worklist.hh](#).

Index

- ~container_base
 - container_base, 8
- ~neighbor_track
 - neighbor_track, 26
- ~suretest
 - suretest, 38
- ~voronoi_cell_base
 - voronoi_cell_base, 40
- add_list_memory
 - container_base, 8
- add_memory_vertices
 - neighbor_none, 22
 - neighbor_track, 27
- add_memory_vorder
 - neighbor_none, 22
 - neighbor_track, 27
- add_particle_memory
 - container_base, 8
- add_vertex
 - voronoi_cell_base, 41
- add_wall
 - container_base, 8
- allocate
 - neighbor_none, 22
 - neighbor_track, 27
- allocate_aux1
 - neighbor_none, 22
 - neighbor_track, 27
- ax
 - container_base, 16
- ay
 - container_base, 16
- az
 - container_base, 16
- bx
 - container_base, 16
- by
 - container_base, 16
- bz
 - container_base, 16
- cell.cc, 61
- cell.hh, 61
 - voronoi_cell, 62
 - voronoi_cell_neighbor, 62
- check_duplicates
 - voronoi_cell_base, 41
- check_facets
 - neighbor_none, 23
 - neighbor_track, 27
 - voronoi_cell_base, 41
- check_relations
 - voronoi_cell_base, 42
- clear
 - container_base, 8
- clear_max
 - radius_mono, 31
 - radius_poly, 34
- co
 - container_base, 16
- compute_all_cells
 - container_base, 8
- compute_cell
 - container_base, 9
- compute_cell_sphere
 - container_base, 9, 10
- config.hh, 62
 - fpoint, 64
 - init_3_vertices, 64
 - init_delete2_size, 64
 - init_delete_size, 64
 - init_facet_size, 64
 - init_marginal, 64
 - init_n_vertices, 64
 - init_vertex_order, 64
 - init_vertices, 64
 - init_wall_size, 65
 - large_number, 65
 - max_delete2_size, 65
 - max_delete_size, 65
 - max_marginal, 65
 - max_n_vertices, 65
 - max_particle_memory, 65
 - max_vertex_order, 65
 - max_vertices, 65
 - max_wall_size, 66
 - tolerance, 66
 - tolerance2, 66
 - VOROPP_VERBOSE, 63
- construct_relations
 - voronoi_cell_base, 42
- container
 - container.hh, 67
- container.cc, 66
- container.hh, 66

- container, 67
- container_poly, 67
- container_base, 5
 - ~container_base, 8
 - add_list_memory, 8
 - add_particle_memory, 8
 - add_wall, 8
 - ax, 16
 - ay, 16
 - az, 16
 - bx, 16
 - by, 16
 - bz, 16
 - clear, 8
 - co, 16
 - compute_all_cells, 8
 - compute_cell, 9
 - compute_cell_sphere, 9, 10
 - container_base, 8
 - container_base, 8
 - count_all_faces, 10
 - current_wall_size, 17
 - draw_cells_gnuplot, 11
 - draw_cells_pov, 11
 - draw_particles, 11
 - draw_particles_pov, 12
 - hx, 17
 - hxy, 17
 - hxyz, 17
 - hy, 17
 - hz, 17
 - id, 17
 - import, 12
 - initialize_voronoicell, 13
 - mask, 17
 - mem, 18
 - mrad, 18
 - mv, 18
 - nx, 18
 - nxy, 18
 - nxyz, 18
 - ny, 18
 - nz, 18
 - p, 18
 - packing_fraction, 13
 - point_inside, 13
 - point_inside_walls, 14
 - print_all, 14
 - print_all_neighbor, 14, 15
 - put, 15
 - radius, 19

- region_count, 15
- s_end, 19
- s_size, 19
- s_start, 19
- sl, 19
- store_cell_volumes, 15
- sum_cell_volumes, 16
- sz, 19
- wall_number, 19
- walls, 19
- xperiodic, 20
- xsp, 20
- yperiodic, 20
- ysp, 20
- zperiodic, 20
- zsp, 20
- container_poly
 - container.hh, 67
- copy
 - neighbor_none, 23
 - neighbor_track, 27
- copy_aux1
 - neighbor_none, 23
 - neighbor_track, 27
- copy_aux1_shift
 - neighbor_none, 23
 - neighbor_track, 27
- copy_pointer
 - neighbor_none, 23
 - neighbor_track, 28
- copy_to_aux1
 - neighbor_none, 23
 - neighbor_track, 28
- count_all_faces
 - container_base, 10
- current_delete2_size
 - voronoicell_base, 49
- current_delete_size
 - voronoicell_base, 49
- current_vertex_order
 - voronoicell_base, 49
- current_vertices
 - voronoicell_base, 49
- current_wall_size
 - container_base, 17
- cut_cell
 - wall, 53
 - wall_cone, 54
 - wall_cylinder, 56
 - wall_plane, 58
 - wall_sphere, 60

- cut_cell_base
 - wall_cone, [54](#)
 - wall_cylinder, [56](#)
 - wall_plane, [58](#)
 - wall_sphere, [60](#)
- cutoff
 - radius_mono, [31](#)
 - radius_poly, [34](#)
- draw_cells_gnuplot
 - container_base, [11](#)
- draw_cells_pov
 - container_base, [11](#)
- draw_gnuplot
 - voronoicell_base, [42](#)
- draw_particles
 - container_base, [11](#)
- draw_particles_pov
 - container_base, [12](#)
- draw_pov
 - voronoicell_base, [43](#)
- draw_pov_mesh
 - voronoicell_base, [43](#), [44](#)
- ed
 - voronoicell_base, [49](#)
- facet_statistics
 - voronoicell_base, [44](#)
- facets
 - voronoicell_base, [44](#), [45](#)
- fatal_error, [20](#)
 - fatal_error, [21](#)
 - fatal_error, [21](#)
- fpoint
 - config.hh, [64](#)
- hx
 - container_base, [17](#)
- hxy
 - container_base, [17](#)
- hxyz
 - container_base, [17](#)
- hy
 - container_base, [17](#)
- hz
 - container_base, [17](#)
- id
 - container_base, [17](#)
- import
 - container_base, [12](#)
 - radius_mono, [32](#)
 - radius_poly, [35](#)
- inc
 - voropp_loop, [51](#)
- init
 - neighbor_none, [23](#)
 - neighbor_track, [28](#)
 - radius_mono, [32](#)
 - radius_poly, [35](#)
 - suretest, [38](#)
 - voronoicell_base, [45](#)
 - voropp_loop, [51](#)
- init_3_vertices
 - config.hh, [64](#)
- init_delete2_size
 - config.hh, [64](#)
- init_delete_size
 - config.hh, [64](#)
- init_facet_size
 - config.hh, [64](#)
- init_marginal
 - config.hh, [64](#)
- init_n_vertices
 - config.hh, [64](#)
- init_octahedron
 - neighbor_none, [23](#)
 - neighbor_track, [28](#)
 - voronoicell_base, [45](#)
- init_test
 - voronoicell_base, [45](#)
- init_tetrahedron
 - neighbor_none, [23](#)
 - neighbor_track, [28](#)
 - voronoicell_base, [45](#)
- init_vertex_order
 - config.hh, [64](#)
- init_vertices
 - config.hh, [64](#)
- init_wall_size
 - config.hh, [65](#)
- initialize_voronoicell
 - container_base, [13](#)
- ip
 - voropp_loop, [52](#)
- jp
 - voropp_loop, [52](#)
- kp
 - voropp_loop, [52](#)
- label_facets
 - neighbor_none, [24](#)

- neighbor_track, 28
- voronoicell_base, 46
- large_number
 - config.hh, 65
- mask
 - container_base, 17
- max_delete2_size
 - config.hh, 65
- max_delete_size
 - config.hh, 65
- max_marginal
 - config.hh, 65
- max_n_vertices
 - config.hh, 65
- max_particle_memory
 - config.hh, 65
- max_vertex_order
 - config.hh, 65
- max_vertices
 - config.hh, 65
- max_wall_size
 - config.hh, 66
- maxradsq
 - voronoicell_base, 46
- mem
 - container_base, 18
- mem_size
 - radius_mono, 33
 - radius_poly, 37
- mne
 - neighbor_track, 30
- mrاد
 - container_base, 18
- mv
 - container_base, 18
- ne
 - neighbor_track, 30
- neighbor_none, 21
 - add_memory_vertices, 22
 - add_memory_vorder, 22
 - allocate, 22
 - allocate_aux1, 22
 - check_facets, 23
 - copy, 23
 - copy_aux1, 23
 - copy_aux1_shift, 23
 - copy_pointer, 23
 - copy_to_aux1, 23
 - init, 23
 - init_octahedron, 23

- init_tetrahedron, 23
- label_facets, 24
- neighbor_none, 22
- neighbor_none, 22
- neighbors, 24
- print, 24
- print_edges, 24
- set, 24
- set_aux1, 24
- set_aux2_copy, 24
- set_pointer, 24
- set_to_aux1, 25
- set_to_aux1_offset, 25
- set_to_aux2, 25
- switch_to_aux1, 25
- neighbor_track, 25
 - ~neighbor_track, 26
 - add_memory_vertices, 27
 - add_memory_vorder, 27
 - allocate, 27
 - allocate_aux1, 27
 - check_facets, 27
 - copy, 27
 - copy_aux1, 27
 - copy_aux1_shift, 27
 - copy_pointer, 28
 - copy_to_aux1, 28
 - init, 28
 - init_octahedron, 28
 - init_tetrahedron, 28
 - label_facets, 28
 - mne, 30
 - ne, 30
 - neighbor_track, 26
 - neighbor_track, 26
 - neighbors, 28
 - print, 28
 - print_edges, 29
 - set, 29
 - set_aux1, 29
 - set_aux2_copy, 29
 - set_pointer, 29
 - set_to_aux1, 29
 - set_to_aux1_offset, 30
 - set_to_aux2, 30
 - switch_to_aux1, 30
 - vc, 30
- neighbors
 - neighbor_none, 24
 - neighbor_track, 28
 - voronoicell_base, 46

- nplane
 - voronoicell_base, [46](#), [47](#)
- nu
 - voronoicell_base, [49](#)
- number_of_faces
 - voronoicell_base, [47](#)
- nx
 - container_base, [18](#)
- nxy
 - container_base, [18](#)
- nxyz
 - container_base, [18](#)
- ny
 - container_base, [18](#)
- nz
 - container_base, [18](#)
- p
 - container_base, [18](#)
 - suretest, [38](#)
 - voronoicell_base, [50](#)
- packing_fraction
 - container_base, [13](#)
- perturb
 - voronoicell_base, [47](#)
- plane
 - voronoicell_base, [47](#)
- plane_intersects
 - voronoicell_base, [48](#)
- plane_intersects_guess
 - voronoicell_base, [48](#)
- point_inside
 - container_base, [13](#)
 - wall, [53](#)
 - wall_cone, [55](#)
 - wall_cylinder, [57](#)
 - wall_plane, [59](#)
 - wall_sphere, [61](#)
- point_inside_walls
 - container_base, [14](#)
- print
 - neighbor_none, [24](#)
 - neighbor_track, [28](#)
 - radius_mono, [32](#)
 - radius_poly, [35](#)
- print_all
 - container_base, [14](#)
- print_all_neighbor
 - container_base, [14](#), [15](#)
- print_edges
 - neighbor_none, [24](#)
 - neighbor_track, [29](#)
- voronoicell_base, [48](#)
- pts
 - voronoicell_base, [50](#)
- put
 - container_base, [15](#)
- rad
 - radius_mono, [32](#)
 - radius_poly, [35](#)
- radius
 - container_base, [19](#)
- radius_mono, [31](#)
 - clear_max, [31](#)
 - cutoff, [31](#)
 - import, [32](#)
 - init, [32](#)
 - mem_size, [33](#)
 - print, [32](#)
 - rad, [32](#)
 - radius_mono, [31](#)
 - radius_mono, [31](#)
 - scale, [32](#)
 - store_radius, [33](#)
 - volume, [33](#)
- radius_poly, [34](#)
 - clear_max, [34](#)
 - cutoff, [34](#)
 - import, [35](#)
 - init, [35](#)
 - mem_size, [37](#)
 - print, [35](#)
 - rad, [35](#)
 - radius_poly, [34](#)
 - radius_poly, [34](#)
 - scale, [36](#)
 - store_radius, [36](#)
 - volume, [36](#)
- region_count
 - container_base, [15](#)
- s_end
 - container_base, [19](#)
- s_size
 - container_base, [19](#)
- s_start
 - container_base, [19](#)
- scale
 - radius_mono, [32](#)
 - radius_poly, [36](#)
- set
 - neighbor_none, [24](#)
 - neighbor_track, [29](#)

- set_aux1
 - neighbor_none, 24
 - neighbor_track, 29
- set_aux2_copy
 - neighbor_none, 24
 - neighbor_track, 29
- set_pointer
 - neighbor_none, 24
 - neighbor_track, 29
- set_to_aux1
 - neighbor_none, 25
 - neighbor_track, 29
- set_to_aux1_offset
 - neighbor_none, 25
 - neighbor_track, 30
- set_to_aux2
 - neighbor_none, 25
 - neighbor_track, 30
- sl
 - container_base, 19
- store_cell_volumes
 - container_base, 15
- store_radius
 - radius_mono, 33
 - radius_poly, 36
- sum_cell_volumes
 - container_base, 16
- sure
 - voronoicell_base, 50
- suretest, 37
 - ~suretest, 38
 - init, 38
 - p, 38
 - suretest, 38
 - test, 38
- switch_to_aux1
 - neighbor_none, 25
 - neighbor_track, 30
- sz
 - container_base, 19
- test
 - suretest, 38
- tolerance
 - config.hh, 66
- tolerance2
 - config.hh, 66
- up
 - voronoicell_base, 50
- vc
 - neighbor_track, 30
- volume
 - radius_mono, 33
 - radius_poly, 36
 - voronoicell_base, 48
- voro++.cc, 68
- voro++.hh, 68
- voronoicell
 - cell.hh, 62
- voronoicell_base, 38
 - ~voronoicell_base, 40
 - add_vertex, 41
 - check_duplicates, 41
 - check_facets, 41
 - check_relations, 42
 - construct_relations, 42
 - current_delete2_size, 49
 - current_delete_size, 49
 - current_vertex_order, 49
 - current_vertices, 49
 - draw_gnuplot, 42
 - draw_pov, 43
 - draw_pov_mesh, 43, 44
 - ed, 49
 - facet_statistics, 44
 - facets, 44, 45
 - init, 45
 - init_octahedron, 45
 - init_test, 45
 - init_tetrahedron, 45
 - label_facets, 46
 - maxradsq, 46
 - neighbors, 46
 - nplane, 46, 47
 - nu, 49
 - number_of_faces, 47
 - p, 50
 - perturb, 47
 - plane, 47
 - plane_intersects, 48
 - plane_intersects_guess, 48
 - print_edges, 48
 - pts, 50
 - sure, 50
 - up, 50
 - volume, 48
 - voronoicell_base, 40
 - voronoicell_base, 40
- voronoicell_neighbor
 - cell.hh, 62
- voropp_loop, 50

- inc, [51](#)
- init, [51](#)
- ip, [52](#)
- jp, [52](#)
- kp, [52](#)
- voropp_loop, [51](#)
- voropp_loop, [51](#)
- VOROPP_VERBOSE
- config.hh, [63](#)
- wall, [52](#)
 - cut_cell, [53](#)
 - point_inside, [53](#)
- wall.cc, [68](#)
- wall.hh, [68](#)
- wall_cone, [53](#)
 - cut_cell, [54](#)
 - cut_cell_base, [54](#)
 - point_inside, [55](#)
 - wall_cone, [54](#)
 - wall_cone, [54](#)
- wall_cylinder, [55](#)
 - cut_cell, [56](#)
 - cut_cell_base, [56](#)
 - point_inside, [57](#)
 - wall_cylinder, [56](#)
 - wall_cylinder, [56](#)
- wall_number
 - container_base, [19](#)
- wall_plane, [57](#)
 - cut_cell, [58](#)
 - cut_cell_base, [58](#)
 - point_inside, [59](#)
 - wall_plane, [58](#)
 - wall_plane, [58](#)
- wall_sphere, [59](#)
 - cut_cell, [60](#)
 - cut_cell_base, [60](#)
 - point_inside, [61](#)
 - wall_sphere, [60](#)
 - wall_sphere, [60](#)
- walls
 - container_base, [19](#)
- worklist.cc, [69](#)
- worklist.hh, [69](#)
- xperiodic
 - container_base, [20](#)
- xsp
 - container_base, [20](#)
- yperiodic
- container_base, [20](#)
- ysp
 - container_base, [20](#)
- zperiodic
 - container_base, [20](#)
- zsp
 - container_base, [20](#)