Voro++: A three-dimensional Voronoi cell library in C++

Chris H. Rycroft
Department of Mathematics, Lawrence Berkeley National Laboratory
Department of Mathematics, University of California, Berkeley

ERNEST ORLANDO LAWRENCE
BERKELEY NATIONAL LABORATORY

Introduction

The Voronoi tessellation was proposed over a century ago, and today it has applications in many scientific disciplines [1]. In physics and materials science, it has been used extensively in the analysis of particle systems, for tracking changes in density, or for examining local neighbor relationships. While several mature software libraries exist (particularly *Qhull* [2], built into *MATLAB*), *Voro++* has been designed specifically for handling large-scale 3D research problems, where flexibility is required.

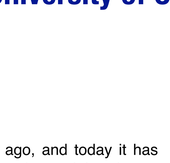
Whereas some codes compute the Voronoi tessellation as a single mesh, *Voro++* makes use of a direct method of calculation whereby each cell is computed individually [3]. This perspective is useful in many physical applications, which often rely on cell-based statistics. It provides a high degree of flexibility, since the computation of each cell can be individually tailored to account for non-standard boundary conditions. Since each cell is computed independently, it is straightforward to generalize to a multi-core architecture and achieve very high parallel efficiency.

Voro++ is free, open source software and specifically concentrates on the 3D tessellation. It is written in object-oriented C++, and has been designed to be easily modified and incorporated into other programs; a command-line utility is also available for interfacing with scripting languages. The library is well-documented and numerous examples are provided on the code website that demonstrate the library's features.

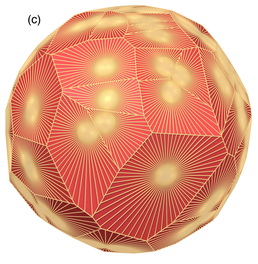
Constructing a single cell

For a set of points in a domain, the Voronoi tessellation can be defined by associating a cell to each particle that corresponds to all of the space which is closer to that particle than any other. For the small 2D particle arrangement shown in (a), the Voronoi cells form irregular convex polygons around each particle. The sides of each polygon are the perpendicular bisectors between neighboring particles.

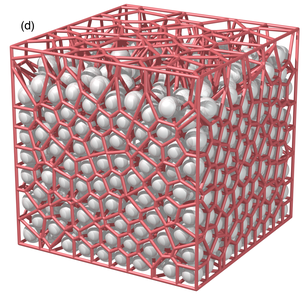
In three dimensions, a Voronoi cell will form an irregular convex polyhedron. One of the main components of *Voro++* is the **voronoicell** class, which handles all the routines for constructing a single 3D cell. The cell is represented as a list of vertices that are connected by a table of edges, and a sample representation is shown in (b). The voronoicell class contains a routine called **plane()**, which can re-compute the vertices and edges based on cutting the cell by a single plane. A Voronoi cell can be built by repeatedly cutting the cell by planes corresponding to the perpendicular bisectors of neighboring particles. Since the cell is convex and only a few vertices need to be recomputed for each plane, this operation runs very quickly.



In a typical case such as (b), where a cell is created by random plane cuts, each vertex will have order 3. However, if a plane intersects an existing vertex, a higher order vertex could potentially form. Without recognizing these cases, inaccuracies in floating point arithmetic can lead to algorithm errors. To handle this, if a plane is within a small tolerance of an existing vertex, *Voro++* recomputes the cell as if that vertex was exactly on the plane, constructing high order vertices as necessary. The code dynamically allocates memory for high order vertices, and can handle very complex test cases such as (c). Here, a large number of plane cuts were applied radially around specific points, to create many vertices of order 64. While high order vertices rarely form in practice, having this capability makes the Voronoi calculations very robust, even when dealing with millions of cells.



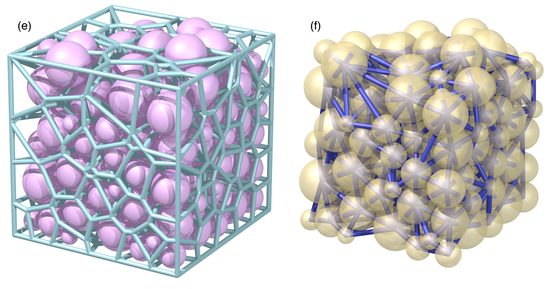
Calculating a tessellation



Voro++ has a class called **container**, which represents a complete simulation region that can be filled with particles. For computational efficiency, the container is divided into a rectangular grid of blocks, each of which store the particles in their part of the simulation. To compute the Voronoi tessellation, the code constructs a voronoicell class for each particle, and builds the cell by applying plane() calls to account for neighbors. It first applies plane cuts for all neighboring particles in the same container block, and then sweeps outwards using a pre-computed list of nearby blocks. The cell calculation is completed once the remaining container blocks are too far away to possibly influence it. The results can be output in a variety of formats, and numerous routines exist for carrying out cell-based calculations, such as computing cell volume. The above image shows a sample packing of 1000 particles in a cube, that is one of the standard code examples. Mixed periodic/non-periodic boundaries are also supported.

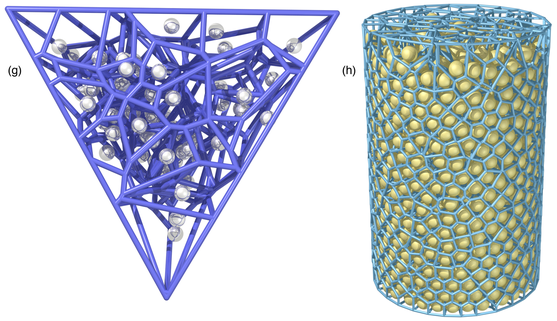
On a Mac Pro with a 2.66 GHz Dual-Core Intel Xeon processor, with standard compilation options, computing the tessellation for 100,000 random particles in a unit cube takes 3.11 s. For comparison, *Qhull* reports a calculation time of 8.58 s for the same test on the same system.

Additional code features



The library can also carry out the radical Voronoi tessellation, which can be used in the analysis of polydisperse particle arrangements, where the planes between Voronoi cells are weighted by the particle radii. A sample tessellation for a packing of 159 polydisperse particles in a cube is shown in (e). In addition, the code can carry out neighbor calculations, so that during the Voronoi cell construction, each plane of the cell is labeled with the ID number of the neighboring particle that created it. Image (f) was created using this information, drawing a line between every pair of particles whose Voronoi cells share a face. The radical Voronoi tessellation and neighbor-tracking Voronoi cell are carried out using two classes called **container_poly** and **voronoicell_neighbor**, which are variants of the standard container and voronoicell classes. To keep the source code as concise and maintainable as possible, the majority of the library is written using C++ templates, and class variants are then created as different template instances. Since the class variations are in-lined during compilation, this results in very high performance. Further extensions have been developed in a nanosphere systems study [4].

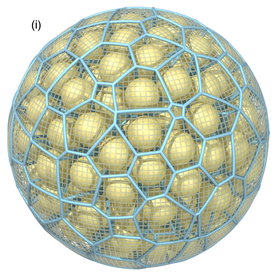
Walls



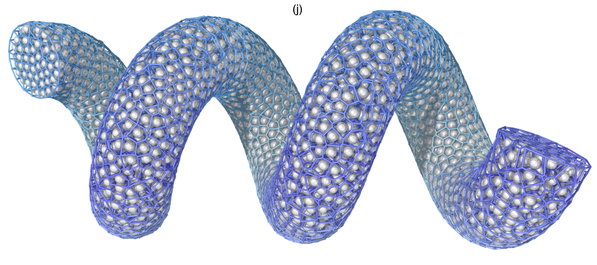
The direct cell-by-cell construction makes it particularly easy to handle non-standard boundary conditions, since the Voronoi cells can be individually tailored by applying additional plane cuts. *Voro++* has a general mechanism for handling these cases, by creating wall classes that can be added to a container. The library contains classes for conical, spherical, cylindrical, and plane walls with arbitrary orientations, and custom classes can be written for other geometries.

In (g), a Voronoi tessellation has been constructed for 64 particles in a tetrahedron created using four planar walls. In (h), a Voronoi tessellation is shown for a cylindrical packing of 2,300 particles generated using a discrete-element method (DEM) pouring simulation. For each cell, the curved cylindrical wall is approximated using a single plane cut that is locally aligned with the cylinder surface. Although some approximation errors can be seen near the top of the packing, this approach generally works well for dense particle packings and walls with low curvature; in the example above, the total Voronoi cell volume is 0.32% more than the exact cylinder volume.

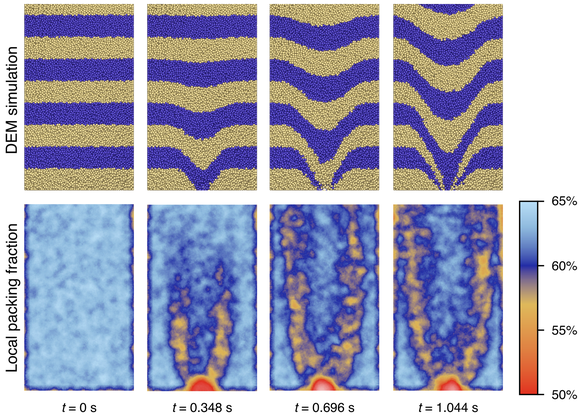
Extra wall accuracy can be achieved by approximating curved wall surfaces with multiple plane cuts. In (i), a Voronoi computation is carried out for a small packing of 262 particles inside a sphere. The curved wall surface is approximated with a fine rectangular grid of plane cuts. The sum of the Voronoi cell volumes differs from the exact sphere volume by 0.039%, and by refining the mesh further, any desired level of accuracy can be achieved.



Since the voronoicell class is designed to only represent convex polyhedra, carrying out computations in non-convex simulation regions may pose some problems, as this could give rise to non-convex Voronoi cells. One approach is to decompose the domain into several convex pieces and carry out a Voronoi tessellation in each. Alternatively, for non-convex walls with low curvature, such as the helix shown in (j), accounting for the wall with a single plane cut to each cell may give acceptable results.

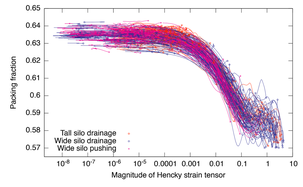


Application to granular flow



Voronoi cells have been used in the study of flowing granular materials, as a way of tracking small changes in packing fraction [5]. The top row of images are snapshots of a DEM granular drainage simulation. The two colors of particles are physically identical and are used to highlight the deformation that takes place as drainage occurs. The bottom row of images show corresponding plots of the local, instantaneous packing fraction. Initially, the packing fraction is approximately 64% everywhere, near to random close packing (RCP). However, during drainage the packing fraction decreases by approximately 5% in the regions of highest strain. It is not possible to accurately track such small changes if the packing fraction is computed just by summing up the number of particles in a local region, as the sample sizes are too small. Instead, the images shown here were computed by dividing the particle volume by the Voronoi cell volume in a local region, which gives much better results.

Tracking small local changes in packing fraction is important in studying dense granular rheology. The graph shown below is taken from a recent publication [6] showing local phase space tracers of strain against packing fraction for a variety of simulations, which is in agreement with the above images. Voronoi volumes have also been used in tracking small porosity changes in pebble-bed reactor simulations [7].



References

[1] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu, *Spatial tessellations: concepts and applications of Voronoi diagrams*, John Wiley & Sons, Inc., New York, NY, 2000.

[2] *Qhull code for convex hull, Delaunay triangulation, Voronoi diagram, and halfspace intersection about a point.* http://www.qhull.org/

[3] Chris H. Rycroft, *Voro++: a three-dimensional Voronoi cell library in* C++ (February 2, 2009). Lawrence Berkeley National Laboratory. Paper LBNL-1432E. http://repositories.cdlib.org/lbnl/LBNL-1432E

[4] Carolyn L. Phillips, Christopher R. Iacovella, and Sharon C. Glotzer, *Stability of the double gyroid phase to nanoparticle polydispersity in polymer tethered nanosphere systems*, preprint, 2009.

[5] Chris H. Rycroft, *Multiscale modeling in granular flow*, Ph.D. thesis, Massachusetts Institute of Technology, 2007. http://math.berkeley.edu/~chr/publish/phd.html

[6] Chris H. Rycroft, Ken Kamrin, and Martin Z. Bazant, *Assessing continuum relationships in simulations of dense granular flow*, accepted in J. Mech. Phys. Solids.

[7] Chris H. Rycroft, Gary S. Grest, James W. Landry, and Martin Z. Bazant, *Analysis of granular flow in a pebble-bed nuclear reactor*, Phys. Rev. E **74** (2006), 021306.

Visit http://math.lbl.gov/voro++/ to download the latest version, and see complete documentation and examples.

This work was supported by the Director, Office of Science, Computational and Technology Research, U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Email enquiries to chr@alum.mit.edu.