SPNC - Sphere Packing Network Construction

Introduction

The `main()` function has been made deliberately verbose to show the steps that SPNC takes in constructing the contact network and pore network. These steps are explained in detail below. The source code is also commented throughout to complement this walkthrough and explain the steps in more detail. The Google `C++` Style Guide is followed as much as possible. The author is by no means an expert in `C++` so a more experienced developer will most likely spot many potential improvements.

Glossary

- **Handle** - A `CGAL` concept similar to a pointer.
- **Vertex** - A point in the Delaunay triangulation
  - **Infinite vertex** - An auxiliary vertex to create an infinite facet/cell with
- **Facet** - A triangle on a tetrahedron in the Delauany triangulation
  - **Infinite facet** - An auxiliary facet defined by two finite vertices (on the hull of the triangulation) and an infinite vertex
- **Cell** - A tetrahedron in the Delaunay triangulation
  - **Infinite cell** - A auxiliary cell defined by three infinite facets and one finite facet
- **Range tree** - A tree data structure for fast point retrieval
- **Disjoint set** - A tree data structure to keep track of merged cells
- **Node** - A single element or point with some properties
  - **Network node** - An element with zero or more edges to other nodes in a network structure
  - **Tree node** - An element with a single 'parent' node and zero or more 'children'
- **Edge**
  - **Network edge** A link between two network nodes
  - **Cell edge** A term used for the connection between two cells, if they share a facet

Assumptions

- Particles can overlap, but do not deform (i.e. remain spherical).
- The assembly is subsampled, where the subsample includes any particles that are fully contained, or partially contained (to any extent) within the subsample limits (see step 2.2).

Walkthrough

The numbered steps in this section correspond directly to the steps in `main.cpp`. For a quick overview of command line usage, type `./spnc --help`. This will display:

```
Usage: ./spnc [-option] [argument]

Options:
-h [--help]                    Show help information
-f [--file_path]               Path to file with a directory list
-p [--porosity_threshold]      Areal porosity merging threshold
-n [--no_particle_overlap]     Are particles assumed to overlap? (0 = yes, 1 = no)
Example: ./spnc -f ../docs/example/dir_list.txt -p 0.4 -n 0
```

1. Input setup

1.1 Read in the command-line arguments

The main hyperparameter in the construction of the pore network is the merging threshold. This threshold essentially controls how "void" a facet should be before it is considered a throat. In step 7.3, SPNC calculates (void facet area/total facet area). If the result exceeds the merging threshold, two cells are merged.

SPNC is build to loop over a collection of assemblies of spheres and construct the contact network and pore network for each assembly. The list of paths to the source directories of those packings is read from the text file `dir_list.txt` and stored in the vector `path_list`. Every line in the text file should contain one path. The directory the path is pointing to should be in the following format:

```
 --- assembly_1
|   |
|    --- DEM
|   |   |   subsample.txt
|    --- networks
 --- assembly_2
|    --- DEM
|   |   |   subsample.txt
|    --- networks
| ...
```

See also the example in `docs/example/assembly_1`. The file `subsample.txt` in the sub-directory `/DEM` is assumed to contain the sphere assembly details. An example is provided in the `docs/example/assembly_1/DEM` directory to show the format the data should be organized in (see also step 2.2). The output of SPNC will be written to `docs/example/assembly_1/networks`.

2. Initialization

2.1 Create packing in which most data is stored

The Pack class is the main interface between `main` and all other classes. It contains all representations of the assembly, including the particle data, triangulation, networks, disjoint-set for merging and range tree for point retrieval.

2.2 Read input data from text file

SPNC expects the particle data to be located a file `subsample.txt` as outlined in step 1.3. An example file is provided in the `docs/example/assembly_1/DEM` directory to show the format of the input data. Subsampling from a discrete-element simulation is generally recommended to remove wall effects. The pore network construction, in particular, will perform best if the assembly is subsampled, where the subsample includes any particles that are fully contained, or partially contained (to any extent) within the subsample limits. Such a subsample (as opposed to including only particles that are fully contained within the limits) will work well with the FindInteriorCellsAndFacets() function below, and prevent the inclusion of highly elongated tetrahedra on the hull of the triangulation.

2.3 Perform Delaunay triangulation

The Delaunay triangulation tessellates the assembly of particles into a collection of tetrahedra (referred to as *cells*). The particle centroids are the *vertices* of the cells and the four triangles forming the tetrahedron are called *facets*. See also this link.

2.4 Compute range tree of the packing

A range tree segments a collection of points in a tree structure, for fast point retrieval. This range tree stores the vertices of the triangulation, as a pair of coordinates and the handle to the vertex (for retrieval of other information, such as the radius of the corresponding particle).

2.5 Determine interior cells and facets

The Delaunay triangulation will form highly elongated tetrahedra on the hull of the triangulation, which are undesirable for the construction of the pore network. To this end, 'interior' cells and facets are identified, away from the hull.

2.6 Initialize disjoint-set data structure

Although cells in the Delaunay triangulation are a reasonably good first approximation of the nodes in a pore network, some tetrahedra will unnecessarily subdivide pores. As such, SPNC checks if each pair of adjacent cells should be merged. To keep track of the merging, a disjoint set data structure is used. Each disjoint set (a tree structure) in the DisjointSet class represents a collection of one or more merged cells.

3. Construct contact network

3.1 Add nodes to the contact network

Construction of the contact network is relatively straightforward. Particle centroids are assigned nodes. Node properties include the particle surface area, particle volume and whether the node is located in the top, middle or bottom of the network. For the particle volume, any overlap with other particles is subtracted. Moreover, any volume of the particle that extends beyond the subsample limits is also removed. The unique identifier of a node in the contact network is the handle of the vertex in the triangulation.

3.2 Add edges to the contact network, and compute edges weights

An edge is assigned between a pair of nodes if the corresponding particles overlap. The range tree class is used identify these overlapping particles. The only edge property used (at the time of writing) is the contact area.

4. Compute cell parameters

4.1 Compute the void volume and surface area in each tetrahedron

In preparation for the construction of the pore network, the void volume and surface area in each tetrahedron are calculated. For the surface area, only an analytical function is available and any particles intersecting the tetrahedron (other than the four particles at the tetrahedron's vertices) are ignored. For the void volume, an an analytical approach is also available, but not used unless particles do not overlap and no spheres intersect a particular cell other than the four spheres located at the four vertices of the cell (see `Pack::CellParameters()` and the command-line option for `no_particle_overlap`). Otherwise, SPNC uses voxels to approximate the void volume in a tetrahedron, accounting for both particle overlap and particles intersecting the tetrahedron.

5. Rotate vertices

While looping over the interior facets, the first steps in checking the merging criterion is to rotate the facet vertices to a horizontal plane. The same transformation is also applied to any other ('nearby') vertices whose particles may intersect the facet.

5.1 Set the facet plane and horizontal plane to rotate to

The three vertices located at the facet define a plane, which is initialized in the Transform class in this step.

5.2 Compute the rotation matrix needed to rotate vertices

Next, the angle and axis of rotation between the horizontal plane and the facet plane are computed. From this, the 3x3 rotation matrix is derived.

5.3 Rotate the facet vertices and nearby vertices to a horizontal plane

Finally, the coordinate vector of each vertex in `facet_vertices` and `nearby_vertices` is multiplied with the rotation matrix to rotate the vertices.

6. Intersect facet

The `Intersect` class contains all functions to intersect the facet under consideration with any particles corresponding to the vertices in `facet_vertices` and `nearby_vertices`.

6.1 Initialize the intersection by providing the facet vertices

To initialize the intersection, the facet is converted to a polygon.

6.2 Set vertices to be considered for intersection with the facet

Next, the nearby vertices are stored in the intersection class to be considered for intersection with the facet.

6.3 Intersect the circles with the facet

If `nearby_vertices` is empty, that is, if no particles (other than the ones corresponding to `facet_vertices`) intersect the facet, then the void area and solid area of the intersection can be computed analytically. If `nearby_vertices` is not empty, another check is performed to see if any of the particles at the nearby vertices actually intersect the facet. If not, the analytical solution is used. Otherwise, the void area and solid area of the intersection are approximated semi-analyticallly, using a polygon-approximation of the circles (derived from the intersecting particles) that intersect the facet.

7. Check merge criterion

7.1 Retrieve the resulting void area and solid area

In the last step of the facet loop, the void area and solid area of the intersection between the facet and facet/nearby particles are retrieved first.

7.2 Store the void and solid area for later use

These areas will be used for the edge properties of the pore network, so they are stored in the `Pack` class.

7.3 If the porosity on the facet is larger than the threshold, merge the two adjacent cells in the distjoint-set

Lastly, the areal porosity on the facet is computed as the void area / (void area + solid area). If this value is lower than the threshold, this facet considered (physically) to be a throat, i.e. a constriction in the void space. If this value is higher than the threshold, the facet is not a throat and the two tetrahedra adjacent to the facet are assumed to encapsulate the same pore. As such, the two cells are merged using the `DisjointSet`.

8. Construct pore network

8.1 Use the disjoint-set to define the pore network nodes

Each disjoint set of (merged) cells is assigned a node in the pore network. Pore network node properties include the void volume, solid volume, surface area exposed to the void space, location in the assembly (top/middle/bottom) and the pore centroid. The unique identifier of a node in the pore network is the handle of the root cell in the corresponding disjoint set.

8.2 Add pore network edges and compute edge weights

If two cells belong to a different pore network node but share a facet, then an edge is assigned. As such, void area and solid area are summed over all shared facets that make up an edge in the pore network. Moreover, the 'conductance' is computed using a (Hagen-Poiseuille) tube model of the pores and throats. Please refer to our publication (see README) for more details on the conductance.

9. Write results to file

All output is written to file (in the `/networks` sub-directory), including the Delaunay triangulation, networks and cell- and facet-properties.