CS 3824 Final Project. Group work.

1 Motivation

Proteins are cellular workhorses – they perform most of biological functions. Often, protein function includes uptake and release of other small molecules such as oxygen. These enter into the protein via ”channels” or ”pathways” – connected voids in the tightly packed protein structure. Examples of such proteins include hemoglobin that carries oxygen around our bodies, or myoglobin that stores oxygen in muscle cells (it is oxygen-rich myoglobin that makes fresh meet look red). Being able to determine voids and pathways inside proteins is important from both pragmatic (medical) and fundamental (how things work?) reasons.

2 The assignment

Design, implement, and test a basic, simple algorithm that searches out for empty cavities in proteins. It does not have to be the most efficient or elegant one. It just has to work.

3 Definitions

3.1 Protein

A protein can be uniquely specified by (X,Y,Z) coordinates of its every atom i, see any file from the Protein Data Bank (PDB). Each atom is assumed to be a solid sphere of radius $\rho(x_i, y_i, z_i)$. Typically, $1 \leq \rho \leq 2$ (in atomic length units called ”Angstroms”). Proteins are tightly packed globs of atomic spheres, Fig. 1. Neighboring atom spheres can touch and even overlap. In fact, it is this intuitive and physically sound representation – CPK – that is utilized by most freely available visualization codes (e.g. texttrasmol) used to visualize protein structures. For example, carbon atoms are often displayed as a grey spheres, nitrogen are shown as smaller blue spheres. An atom whose name begins with an ”M” would be displayed as a purple or magenta sphere. A typical protein contains between 500 to 3000 atoms.

3.2 Voids and Cavities

A point in space is said to belong to *void* space if a *probe sphere* of specified radius ρ_w can be placed at that point such that the sphere does not overlap with any protein atom. In biology, the relevant $\rho_w = 1.4$ (radius of water molecule). A *cavity* is void space completely enclosed in the protein. That is there is no path of void points that connect any cavity point to the space *outside* the protein.

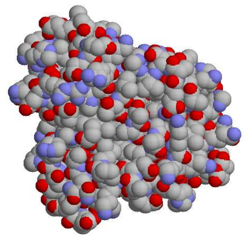
Figure 1: A typical small protein (myoglobin) in the so-called ”CPK” or ”space-fill” representation. Each atom is shown as a sphere of appropriate radius.

4 Specific tasks

Design, test, and implement an algorithm that finds all cavities in the input protein structure. Each cavity point needs to be identified within the accuracy of 0.25 (Å). The code will report a discrete set of spheres representing all of the cavities found in the protein. The spheres that belong to the same connected cavity should be placed no more than 0.25 units from each other along x, y, and z. In other words, you are representing each cavity with a set of spherical ”pixels” at 0.25 resolution. It is OK to miss a cavity, if one needs higher that 0.25 resolution to find it. For example, suppose the probe radius $\rho_w = 1.4$, and the largest sphere that can fit anywhere inside the protein without touching any of the protein atoms (and not connected to the outside) has radius 1.5. This cavity may be missed at 0.25 resolution, but will be identified at 0.05 resolution. However, if the largest sphere that can fit has radius of 1.7, it must be found at 0.25 resolution, and a sphere of radius 1.4 must be placed in its center ± 0.25.

You can also place up to additional 6 spheres around it, distance ±0.25 along x, y, and
z, if none of them touch any of the atoms. The cavity will be represented by this set of spheres. Due to finite resolution the exact number of spheres may differ (which is OK), as long as the cavity is found and represented by the appropriate spheres. Likewise, it is OK to report a false cavity if its elimination requires higher than 0.25 resolution, for example if higher resolution is required to determine that the putative cavity is actually connected to the outside.

4.1 Formats

Your code assumes a very specific input format, called "PDB" (Protein Data Bank). Download the example input and output files. The latter is just a format example, and does not necessarily have all the cavities present. The example is a cooked-up "spherical" protein. The output format has to be exactly the same as in the example. It is extremely important that all the spacings are identical to those in the examples. Field 2 is atom's number i. The first letter of field 3 is the name of the chemical element of that atom, e.g. "S" for sulfur (visualized as a yellow sphere). Field 4 is a 3-letter code for amino-acid name, field 5 is its number. You do not have to worry about those. Fields 6,7,8 are x_i, y_i, z_i coordinates of atom i of the protein or the sphere representing a cavity. Feel free to put 0.00 in the last but one field. This number is irrelevant. The last field is ρ_i. It varies from atom to atom: a typical value for carbon is 1.7, hydrogen 1.2, etc. All spheres representing cavity points must have the same $\rho = \rho_w$. The index for cavity spheres, field 2, must start at $\imath = 10001$ to distinguish them from the protein atoms. The index in field 5 must start at 5001. "CAV" in the 4th field indicates that the given sphere belongs to a cavity. Also, in the output file, the cavity records are always appended to the end of the input protein file, see example output.

4.2 The code

Strictly programming guidelines, see below. C, C++, or PERL is preferred. Along with a project report, you will submit your code (If you must use Java: submit .jar in addition to the source). Ample comments are a must. Do not optimize the code for speed, but make sure it produces an answer within an hour at most for structures with up to 2000 atoms. It should give correct answer for a few, relatively simple test structures that we will use to test it. We will test your code first for $\rho_w = 1.4$, but it should also give meaningful answers for other values of probe ρ within $1 \leq \rho \leq 5$ interval. As well as for lower resolution such as 0.5. The code takes only three input parameters, and outputs just one file in the exact format as the example output you have. `mycode -i inputfile -o outputfile -probe 1.7 -resolution 0.25`

4.3 What to report. Project stages

1. Stage 1. 50 points. Date to be announced. During the first stage, each group gives a 10 min. in-class presentation to outline their plan. Followed by a 5 min. discussion with the rest of the class. This is your chance to get a feedback to see if you are on a right track. Time limit strictly enforced. Focus on general strategy. No algorithm or implementation details are expected at this stage.

2. Stage 2. 100 points. Due date to be announced. Each groups submits a typed progress report. Up to 2 pages, 12 pt font. It must describe the algorithm you are going to implement (pictures that illustrate how the algorithm works are a must), ideas on how to test it, etc. You must describe exactly how you will distinguish cavity from void. If you already have a code at that stage (which is encouraged), give the results. But don't submit the code.

3. Stage 3. 250 points. Final typed report. By the end of class. No exceptions. Exact due date to be announced. No late reports under any circumstances. The report, up to 3 pages long (12 pt font) must include a careful description of the algorithm (with schematics and diagrams), an outline of the implementation, and test results. Pictures are a must. You also submit your code which we will compile and test. If the code does not pass simple test cases, including if we can not visualize your output because you did not adhere to the format specs, you are going to lose a lot of points.

Submission Each group submits one PDF document, with roles of each partner clearly indicated. Everyone in the group will get the same score, assuming roughly equal amount of effort by group members. Otherwise, if the apparent amount of effort is highly unequal, the individual scores will be scaled accordingly.

Programming guidelines

1. Your program should be submitted as a single zip or tar archive.
2. The archive must include a README with a clear compilation instructions. Those should be extremely simple. You should assume that we will be testing your code on a standard UNIX platform, e.g. CentOS 6.
3. The name of the archive should be: last-name-of-student-1.last-name-of-student-N.ProjectCS3824.zip
4. In the header of each class, you are required to include the following information about the assignment: (a) Name of IDE or compiler used. Use very standard ones. e.g. g++ (b) Full names of all partners.
5. Use only standard libraries in your solutions.