**Protein Surface Model Generator**

System Documentation

Alex Gawronski

February 20, 2010

Table of Contents

[Introduction: 3](#_Toc254441169)

[Requirements: 3](#_Toc254441170)

[Installation: 3](#_Toc254441171)

[Running: 3](#_Toc254441172)

[Obtaining Results: 3](#_Toc254441173)

[Implementation: 4](#_Toc254441174)

[Code Style: 4](#_Toc254441175)

[Preliminary Script: 4](#_Toc254441176)

[Finding Ligand Pockets: 4](#_Toc254441177)

[Non-Peptide Ligands: 4](#_Toc254441178)

[Peptide Ligands: 5](#_Toc254441179)

[Electrostatic Potential: 6](#_Toc254441180)

[Molecular Surface Meshes: 7](#_Toc254441181)

[Creation of Notation 3 (N3) File: 7](#_Toc254441182)

[Future Work: 8](#_Toc254441183)

# Introduction:

The Protein Surface Model Generator (PSMG) is an add-on to the molecular visualization program Chimera. Its purpose is to quickly create N3 representations of all proteins in the Protein Data Bank. The representations include spatial information about the shape of the protein annotated with electrostatic potential, amino acid and ligand binding site information.

# Requirements:

PSMG only runs on linux and requires at least 1 GB of memory. Hard drive space will vary on usage, but can be substantial if all proteins are used as input. An average protein will have an N3 output file size of 200MB x 54000 proteins = 10.8 TB.

# Installation:

The PSMG package has everything needed for the program to run. Simply extract the package into a desired directory.

# Running:

To run the script on one PDB structure, edit the "makeMesh" file with the correct PDB ID. The other two parameters are maximum peptide length (# of AA) and ligand pocket distance (angstroms). The first determines which proteins to consider as peptides and the second is determines how far from the ligand to look for a pocket. This file can then be executed through double click or through command line "./makeMesh" or "sudo ./makeMesh". To run all the PDB structures, use the "makeAllMeshes" executable in a similar fashion.

Obtaining Results:

Currently the output of the script is generated in "./chimera/bin" as a N3 file. If the file does not exist after running refer to “ErrorLog” for information on why the process failed.

# Implementation:

## Code Style:

The majority of the code is within ``makeMeshes.py`` in the surface color package (Chimera/share/SurfaceColor). The reason for this is because the procedure for coloring surfaces traverses all the surface faces, similar to PSMG which must traverse all the faces to annotate data. The code itself is a mix of “runCommand()” statements as well as direct manipulation of the Chimera model. “runCommand()” allows for calling any command line function from the code providing a method of controlling Chimera at runtime. While these statements invoke Chimera to do various tasks the model is accessed to retrieve the calculated data. Most steps in the procedure follow this kind of pattern.

## Preliminary Script:

Before the surface models can be built there is some necessary preparation that must be done. This includes retrieving all the four-character protein ids from the PDB, parsing these IDs, and running the visualization program. The first step of obtaining the PDB ids was accomplished using a short Hypertext Preprocessor (PHP) script. The parameters of this script are boolean flags for which types of structures to obtain from the PDB: protein, DNA, RNA or hybrid. This is useful even though the scope of this project is only proteins, it may be extended to include all structures. When the script runs it creates an XML query which is then sent to <http://www.rcsb.org/pdb/rest/search>. When the results return they are echoed, which can then be piped to a file. In the case of this project this is a file of all the protein ids. This file is then parsed using a simple Linux script which runs each protein id as an argument for the Chimera. The actual PDB file is then downloaded by Chimera by calling the “open” command. Also at this step all solvents are removed through the “delete” command.

## Finding Ligand Pockets:

### Non-Peptide Ligands:

This procedure identifies all the ligand pockets in a protein. The algorithm is composed of three components. The algorithm begins with retrieving the Chimera molecule model to obtain the residue list. This list is then looped through checking each residue for a property called “isHet”. This property is true if the residue is nonstandard, or heterogeneous, which means different from the norm. However since the residue list contains the ligands as well, and many ligands are not standard residues, they are put into the heterogeneous category as well. This way specific ligands are obtained so they can be worked with individually. This is crucial since there may be more than one instance of the same kind of ligand in a PDB file.

When one of these “residues” are found, first it is checked to see if it's actually a ligand. There is no actual property in the Chimera residue model to identify it as a ligand. However there is a command called “select” that can select ligands with the parameter “ligand”. The pocket finding procedure requires that the ligand be selected using “select” command. If nothing is selected the procedure will not occur. Therefore if the “select” command was called with the parameters of the residue id and “ligand”, only real ligands will remain selected. The algorithm can run with only the “select” statement to check for ligands, but it is faster to initially filter using “isHet”. Furthermore peptide ligands would fail since they are chains of residues, and require a separate procedure.

With the ligand found, the pocket finding procedure can continue. This is done by using a “zone” parameter in the select command. This will select all residues which are a specified number of angstroms away from the current selection. The zone was set to less than 0.5 angstroms between surfaces since it is the default for finding ligand pockets. Once the command is called, all the residues that make up the pocket should be selected as well. The ligand is still selected at this point so it must be deselected using the “~select ligand” command. From the selection the residue ids can be derived and are stored to memory in a multidimensional array, assigned to the id of the ligand.

### Peptide Ligands:

As mentioned earlier, peptides, which are short amino acid chains, will not work with the first algorithm. They need to be identified and dealt with separately which makes up the next two components of the ligand pocket finding procedure. To identify the peptides, the residue list is traversed. Each residue is checked for what chain they belong to. A “chain” is a part of the tertiary structure of a protein where multiple chains of amino acids are bound together to form the complete protein. The number of residues belonging to each chain is stored in memory within a multidimensional array.

The final component loops through the chains discovered in the previous step. If the chain is less than or equal to 40 (default) residues in length it is deemed a peptide and the pocket finding procedure continues. This value is intentionally high to guarantee the finding of all peptides and it does not increase errors since the peptide is checked with the method mentioned earlier. This procedure is identical to the one used in the previous component with one difference. This procedure deselects residues around non-peptide ligands. This is because some non-peptide ligands are given the same chain ID as a peptide ligand, or a short chain may not be a ligand but will have non-peptide ligands associated with it. With these situations handled, all ligand pockets of a protein are stored in memory ready to be written to an N3 file.

## Electrostatic Potential:

The PSMG package comes with an installation of DelPhi which is used to determine electrostatic potential values. Before DelPhi runs some modifications are made to the protein structure. All these functions are provided by a popular tool called Dock Prep. Dock Prep carries out the following functions:

* Delete solvent
* Delete non-complexed ions
* If alternate atom locations, keep only highest occupancy
* Convert nonstandard residues to standard
* Fix incomplete side chains
* Add hydrogen
* Add charges

Once the preparation step is completed, ligands are deleted and the structure is written to a PDB file. Before DelPhi can run a parameter file must be generated. An example file is shown in table 2. With this parameter file DelPhi has all the required data to solve the Poisson Boltzmann equation. Currently the only way to modify these parameters is to modify the code.

*TABLE 2: DelPhi parameter descriptions and chosen values.*

|  |  |
| --- | --- |
| Parameter | Purpose |
| in(pdb,file="2jfaH.pdb") | The input PDB file of the structure modified be Dock Prep. |
| in(siz,file="amber.siz") | The radii file, list of all possible amino acid separated into atoms and given radii values. [Tsail et al., 1999] (Figure 6) |
| in(crg,file="amber.crg") | The charge file, list of all possible amino acid separated into atoms and given partial charge values [Tsail et al., 1999] (Figure 6) |
| out(phi,file="2jfa.phi") | Output file as a phimap, a grid with electrostatic potential values. |
| indi=2 | Interior dielectric. This is set to two by recommendation of the DelPhi developers for such a large set of proteins (default). |
| exdi=80.0 | Exterior dielectric constant of the solvent (water default) |
| prbrad=1.4 | Radius of probe (water default) |
| salt=0.15 | Salt concentration in mol/L. Value of NaCl in isotonic solution [Scatchard et al., 1938]. |
| ionrad=2.0 | Radius of salt ions (NaCl default) |
| nonint=20 | Number of iterations of the non-linear calculations (full PB equation) |

## Molecular Surface Meshes:

With the ligands found and electrostatics calculated, the surface can now be annotated with this data. Firstly an internal surface model had to be generated. This is invoked in the script using a “surface” command. Chimera uses the Maximal Speed Molecular Surface (MSMS) method to generate the molecular surface. The DelPhi calculations in the form a phimap are imported into Chimera as volumetric data. At this point the “scolor” or surface colouring command is called. This function maps the volumetric data from the phimap to each vertex in the MSMS surface model. Here the code was modified to call a function within the python script to retrieve this mapping. No actual colouring occurs since Chimera runs in no-graphics mode.

The “colorMap” is then passed into the function “printVerticies()” in “makeMeshes.py”. Here the data is organized into a single, easily readable, data structure. The first step is to retrieve all the data from the Chimera MSMS surface model. This should be trivial but the Chimera model separates the surface into “surface pieces” for some internal purpose. Therefore each of these pieces must be retrieved and processed separately. For each piece the vertices are obtained. These vertices are then passed into the “colorMap” to retrieve the volumetric data which are the electrostatic potential values. Each of these vertices is mapped to an atom which is mapped to a residue in the surface model. Now that the residues are known they can be used to access the ligand pocket array. With this procedure complete, the data can be entered into a multidimensional array with one entry per vertex with residue, electrostatic potential and pocket mapped to it.

## Creation of Notation 3 (N3) File:

The data structure created in the first section of “printVerticies()” is traversed to write out all possible triples in N3 format. The first step writes out the information about the current protein. The protein, stored as a molecular entity, is given a solvent excluded surface which is represented by a mesh. The next step is mapping the mesh to all the faces it contains. At the same time all the amino acids that correspond to these faces are created, each with their own solvent excluded surface and mesh. A temporary data structure is created to map which faces correspond to which amino acid and chain. The mapping is then used to print all the amino acid meshes with their corresponding faces. Also at this point any ligand pockets are also written to file. The final step prints each face followed by each vertex that belongs to each face. The vertex triples include position, electrostatic, normal data.

# Future Work:

* Make DelPhi parameters easier to modify by the user
* Modify code so the program can be easily applied to newer versions of Chimera.
* Provide support for DNA, RNA and Hybrid structures
* Add Parallelization
* Dynamically generated protein-specific parameters