Using Structure Tensors to Find Beta Sheets in Protein Volume Images - An Implementation of Yu & Bajaj's 2007 Technique

Jeff Donner

December 6, 2008

1 Overview

It is possible to get 3D volume images of protein density, via methods which are outside the scope of this paper. One helpful way to *identify* arbitrary proteins, is to take a mathematical signature and compare them against a database of similarly derived signatures, to at least narrow down the search for what known proteins the new one might be. The two most prominent structures (known as *secondary structures* within proteins are, alpha helices and beta sheets. Alpha helices are usually medium-to-long straight cylinders of fixed thickness (5 Angstroms). Beta sheets are much less well-defined in shape, including curving, to almost back on themselves. The one thing that is constant is that they are locally flat, and within a certain range of thickness (something like 3.5 Angstroms). This program implements Yu & Bajaj's method [1] of using the *local structure tensor* to detect and find the bounds of beta sheets.

2 Technique and Algorithm

The *structure tensor* is an extension of the Principle Components Analysis (PCA) idea. We might try to get the principal components *of the gradient*, ie the shape, from the *gradient tensor*, all pairwise products of x, y, z directional derivatives

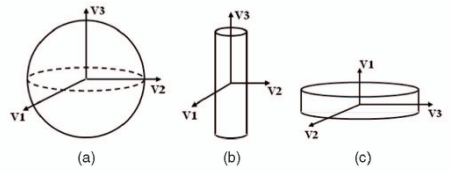
$$G = \begin{pmatrix} f_x^2 & f_x f_y & f_x f_z \\ f_x f_y & f_y^2 & f_y f_z \\ f_x f_z & f_y f_z & f_z^2 \end{pmatrix}$$

and then take the eigenvalues and eigenvectors from this. But, this produces only a single non-zero eigenvector, the net gradient. We want the other two components as well, so we bring in 'influence' from the gradients of neighboring voxels, so that the tensor contains information that reflects a wider area. And, this will also produce up to 3 eigenvectors when analysed. We gather

this 'influence' by convolving with a Gaussian function, g_{alpha}, resulting in the *structure tensor*

$$T_\alpha = \begin{pmatrix} f_x^2 * g_\alpha & f_x f_y * g_\alpha & f_x f_z * g_\alpha \\ f_x f_y * g_\alpha & f_y^2 * g_\alpha & f_y f_z * g_\alpha \\ f_x f_z * g_\alpha & f_y f_z * g_\alpha & f_z^2 * g_\alpha \end{pmatrix}$$

The relative sizes of the eigenvalues indicates the local shape:

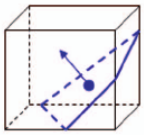


Here v_1 is the direction of greatest gradient, that is, the density drops off quickest in that direction, followed by v_2, v_3.

The algorithm runs as follows:

We start with local density maxima (these points will be usually within alpha helices or beta sheets), then see whether the local structure is characteristic of a beta sheet. This much had been done previously, what was new with this paper is actually *finding the length of probable beta thickness* along the eigenvectors. This is necessary because other, less-dense structures such as coils and turns, show similar curved, flattish local structure, but beta sheets can be distinguished from them by their thickness.

If a point is found likely to be within a beta sheet, we take a plane perpendicular to the predominant eigenvector, intersect it with a small cube around the data point, and use the intersections with the edges as new candidates for being a beta point, thus exploring away from each seed.



When there are no more points to be explored, a triangle mesh is generated from the intersecting polygons, and those comprise the 'beta signature' of the protein.

3 Programming Decisions

ITK [**?**] was chosen because it is pre-made for medical image processing, and has much of what this project needed. It required time to absorb, and the math and methods that it uses are obscured by being abstracted across many layers (ie well-designed from a software engineering standpoint!), but, the author would consider it first, again, for similar projects. Only the gradient tensor and structure tensors themselves were not already present, but they were constructible from other ITK parts without great difficulty.

4 Technical Description

As mentioned, we use ITK, a powerful, free medical imaging toolkit. ITK's architecture is to use image processing *filter*s in which you hook later stages up to earlier stages. There's also a lot of other functionality for things this framework doesn't apply to.

We wrote an .mrc ITK IO file filter, because the only tool we have to generate volumes from protein description files (.pdb) (pdb2mrc, part of the EMAN package) produces the .mrc format.

Our use is: First the

1. Mrc format reader. This was added to our own local version of ITK (not submitted to the toolkit yet, though that is planned.)

2. We find local maxima ourselves (ITK has added this functionality since the project started, but we've continued to use our hand-made version).

3. Per the algorithm, we do a breadth-first search (this was a choice, it could have been depth-first) on each of the seeds for its fit as a beta point, per the equations above. We do this with ITK, a series of filters. For each candidate point, we:

4. Find its physical coordinates. Create a chunk of volume resampled around it, to support the Gaussian operations that come later. (This, because these operations operate on centered pixels, and later coordinates aren't exactly on data points, as the initial seeds are. Resampling makes a new volume, where the chosen coordinate and its neighbors, *are* on data points.)

5. Apply a derivative filter parallel to each axis (ITK's `DerivativeImageFilter`). We make an image rather than an single point from this, to give support to the Gaussian that we'll apply later.

6. Cross-multiply these 3 images (by hand) to form

7. the gradient tensor, where each point is a 9 element matrix

$$G = \begin{pmatrix} f_x^2 & f_x f_y & f_x f_z \\ f_x f_y & f_y^2 & f_y f_z \\ f_x f_z & f_y f_z & f_z^2 \end{pmatrix}$$

   (as it is symmetric actually we need only the 6, upper triangle components).

8. The next step is to blur the information, across corresponding components in G:

$$T_\alpha = \begin{pmatrix} f_x^2 * g_\alpha & f_x f_y * g_\alpha & f_x f_z * g_\alpha \\ f_x f_y * g_\alpha & f_y^2 * g_\alpha & f_y f_z * g_\alpha \\ f_x f_z * g_\alpha & f_y f_z * g_\alpha & f_z^2 * g_\alpha \end{pmatrix}$$

For this we need 6 (taking advantage of symmetry) `itk::GaussianBlurImageFunction`s, each taking one component of our gradient tensor. Since each expects an ordinary, scalar volume, we must give each a `itk::NthElementImageAdaptor`, plugged into one component of the gradient tensor.

9. Then, because the eigen analysis needs to work on the matrices, we write the blurred values right back into the respective components of the Hessian of the original, central volume element, going against the grain of the framework. It should be a bit faster though, and not so against the grain of the framework as to break.

10. We then perform eigen analysis on the structure tensor. (`itk::TotalEigenImageFilter`).

11. We take the eigen values and corresponding eigen vectors, in descending order of eigen values. These, we'll call $\lambda_0, \lambda_1, \lambda_2$.

12. In the order of the.. plane:

    $sheet_{min} <= t_1$ and $t_1 <= sheet_{max}$

    (`constants::BetaMin` and `constants::BetaMax` are the program's names for the constants) and

    $$min\left(\frac{t_2}{t_3}, \frac{t_3}{t_2}\right) > max\left(\frac{t_1}{t_2}, \frac{t_1}{t_3}\right)$$

    For the 2nd formula, recall that in order to be a beta point, thickness t_1 must be significantly smaller than both t_2 and t_3. The intuition of the formula is, starting with the right side, we take the max of the $t1$ ratios to the other, as the *worst interpretation* for being a beta. On the left, we take the *best* interpretation of either of t_2 and t_3, *not* being

    In other words, we take the most antagonistic interpretation of t_1 'sticking out' from the other two, more than the other two stick out from each other.

13. If the thicknesses pass this test, we store this point as a beta point, and explore its neighborhood, in the next step.

14. From here we compute the plane that is perpendicular to the highest eigenvalue that goes through our current point, and calculate where it intersects the edges of our current cell. (Our cell is the resampled voxel, though it could be any reasonable size.) These intersections become the new beta point candidates, and we push them to the back of the queue.

15. We repeat until we find no more beta points. We prevent infinitely fine searching for beta points between others by not processing any candidate that isn't at least 0.5 of a voxel away from all others.

5 Results

PDB	Yu & Bajaj

References

[1] Z. Yu, C. Bajaj. *Computational Approaches for Automatic Structural Analysis of Large Biomolecular Complexes* IEEE/ACM Transactions On Computational Biology And Bioinformatics, Vol. 5, No. 4, October-December 2008

[2] http://www.itk.org