

# Visualizing the Uncertainty in Macro-Molecular Structures

Edward Dale

November 19, 2005\*

---

\*Id: finalPaper.tex 205 2005-11-19 21:21:29Z scompt

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>B-Factors</b>	<b>4</b>
<b>3</b>	<b>Spiegel</b>	<b>4</b>
<b>4</b>	<b>Reading Atoms</b>	<b>4</b>
<b>5</b>	<b>Visualizing Atoms</b>	<b>5</b>
<b>6</b>	<b>Visualizing Backbones</b>	<b>6</b>
<b>7</b>	<b>Visualizing B-Factors</b>	<b>7</b>
<b>8</b>	<b>Transparency</b>	<b>8</b>
<b>9</b>	<b>Filtering Atoms</b>	<b>9</b>
<b>10</b>	<b>A Specialized Atom Camera</b>	<b>10</b>
<b>11</b>	<b>Intersections</b>	<b>11</b>
<b>12</b>	<b>Making Movies</b>	<b>12</b>
<b>13</b>	<b>Conclusion</b>	<b>12</b>
<b>14</b>	<b>Future Work</b>	<b>12</b>

## List of Figures

1	The Spiegel Pipeline . . . . .	4
2	Atoms as red dots . . . . .	5
3	Atoms as red spheres . . . . .	6
4	Atoms with a material and cpk colors . . . . .	6
5	Protein Backbones . . . . .	7
6	Atoms with correct atomic radii . . . . .	8
7	Atoms with b-factors . . . . .	8
8	A frame with transparency . . . . .	9
9	Same frame re-visualized . . . . .	9
10	Unfiltered Protein . . . . .	10
11	Filtered Waters . . . . .	10
12	Filtered Protein . . . . .	10
13	The blocks making up the <i>AtomCam</i> . . . . .	10
14	Atom intersections (~1700) . . . . .	11
15	B-Factor intersections (~4800) . . . . .	11

## 1 Introduction

The goal of this independent study was to visualize proteins in order to determine if their constituent atoms were possibly occupying the same space, a physical impossibility. The tool used for visualization was Spiegel, the advisor was Hans-Peter Bischof, and the domain expert was Paul Craig. This paper will detail the steps taken on the way to determining whether the atoms were intersecting.

## 2 B-Factors

Every atom in every protein in the Protein Data Bank [BWF<sup>+</sup>00] has a certain amount of uncertainty associated with its position. This uncertainty derives from disorder in the crystal that was used to determine the structure and thermal motion in the structure [Rho00]. The goal of this project is to see if, with increasing temperature, the space that an atom can occupy (its b-factor sphere) overlaps with other atoms.

## 3 Spiegel

The tool selected to do the visualization was Spiegel [gra]. It provides a pipeline architecture (Figure 1) that allows one to write blocks that accomplish little tasks and can be connected together to accomplish a greater task. The building blocks ended up contributing a lot to the quick progress of the project by facilitating reuse and encapsulation.

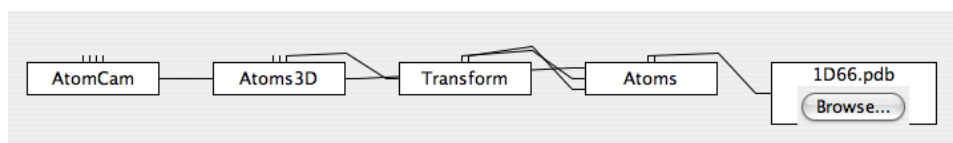


Figure 1: The Spiegel Pipeline

## 4 Reading Atoms

The first step in the Spiegel pipeline and in visualization in general is to get the data to be visualized. This was accomplished using an extractor that just delegated to a class from Jmol [Jmo]. This saved all the work of parsing

and comprehending the .pdb file format. Jmol returns a data structure containing all of the information contained inside the file, which was then trimmed and stored into a `PdbAtom`, containing only the data needed for the project.

The first snag of the project came when these `PdbAtoms` were placed into a `Map`, which doesn't preserve order. The .pdb file format is essentially an ordered list of atoms. When the atoms were stored in an unordered data structure, many assumptions were invalidated and some visualizations broken. This was naturally resolved by storing the atoms into a sorted container (`SortedMap`).

The extractor places the `PdbAtoms` into an `AtomIdMap`, which is then sent through the Spiegel pipeline. The primary reason for using a specialized type of map is that the generics in Java 5 are only available at compile-time. To ensure type-safety throughout the system, these specialized subclasses are necessary.

## 5 Visualizing Atoms

The initial visualization dumped atom information into the existing star data structure and sending it off to the existing star visualizer. This resulted in a bunch of points being displayed that somewhat resembled the correct atom structure (Figure 2). Notice the large spheres on the right that are artifacts of the star visualizer. This validated the atom extraction and encouraged moving on to visualizing the atoms in a more realistic manner.

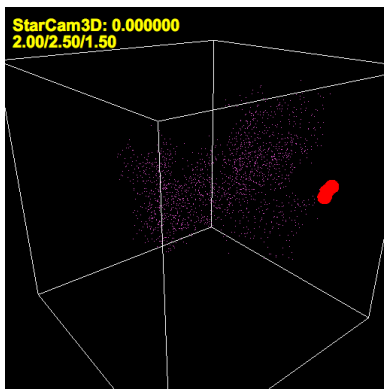


Figure 2: Atoms as red dots

Because of the physical structure of atoms, it was decided to visual-

ize them as spheres. This made the code very easy: iterate through the `AtomIdMap` and create a sphere using the position stored in the `PdbAtom`. At this point, all spheres were created with the same size and the same red color (Figure 3).

At a certain point, the sea of red spheres becomes a red blob. To avoid this, the spheres were given a material and a light was added to the scene, causing the spheres to look like spheres instead of circles. There are atom numerous coloring schemes available. A simplified RasMol cpk color scheme [cpk] was chosen, making the images much easier to interpret (Figure 4). The color scheme only has colors for a dozen atoms, but the majority of the proteins being targeted by the project are built from these atoms.

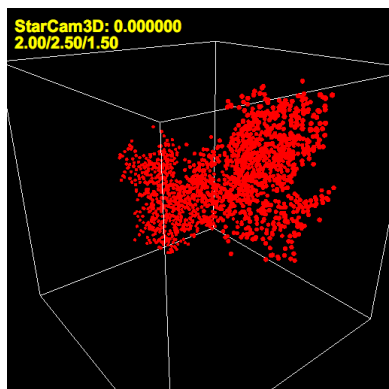


Figure 3: Atoms as red spheres

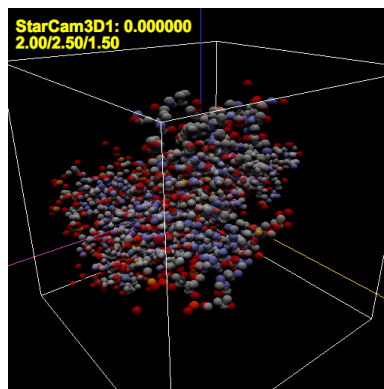


Figure 4: Atoms with a material and cpk colors

## 6 Visualizing Backbones

The second test of the extraction was to try to visualize the backbones of the protein. There can be none to many of these backbones per protein and each one is made up of a chain of carbon atoms called alpha-carbons. To visualize each chain of atoms, a `LineStripArray` was created connecting the atoms (Figure 5).

This is the first place that the ordering problem mentioned in section 4 was encountered. The order of the atoms is very important to the backbones because each `LineStripArray` must be created using the ordered list of the atoms that constitute it. Without a sorted data structure, the chains appeared somewhat correct because they were connected to the proper atoms,

but in the wrong order.

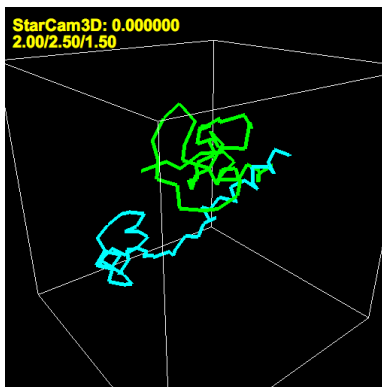


Figure 5: Protein Backbones

There still exist improvements to be made to the backbone visualization. Right now, the colors of the chains is arbitrary, however, there exists a previous color that should probably be used. Also, the visualization might improve with the use of 3D cylinders to visualize the chains as opposed to 2D lines. Splines would be another possible method to connect the atoms, yielding much smoother chains. It might also be advantageous to convert the atom chain block in the Spiegel pipeline to be a generic atom-connecting block.

## 7 Visualizing B-Factors

Before visualizing the b-factor sphere of an atom, the actual size of the atom needed to be taken into consideration. The atomic radii listed at [siz] were used to display correctly scaled atoms (Figure 6). By having the scaling applied only once, right before the image is actually displayed, the visualization routines can draw on the canvas and consider 1 pixel to be equal to 1 angstrom ( $\text{\AA}$ ). This convenience simplifies the blocks at the beginning of the pipeline.

There was a delay getting to this part of the project because a suitable formula for the radius corresponding to an atoms b-factor needed to be found. In the end, the slightly simplified equation 1 was used.  $B$  is the b-factor from the .pdb file and  $u$  is the root mean square displacement of the atom from its rest position. Notice that this equation does not have a temperature component; this is the simplification that was used. A fu-

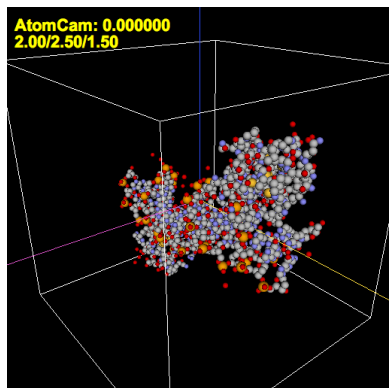


Figure 6: Atoms with correct atomic radii

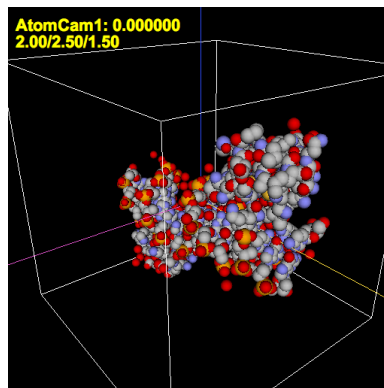


Figure 7: Atoms with b-factors

ture task might be to integrate temperature into the equation to allow the visualization to be tuned and the goal of the project completely realized.

$$B = 8 * \pi^2 * u^2 \quad (1)$$

By solving for  $u$  and plugging in  $B$ , the displacement can be calculated and added to the atomic radius to determine the b-factor sphere. Equipped with the radius, the b-factors could be visualized (Figure 7).

## 8 Transparency

Essentially, the b-factor sphere is an area that the atom could be occupying. The most logical way to visualize this was determined to be transparency. Placing the atom as an opaque sphere inside its transparent b-factor sphere is an attractive possibility. Parameter inputs were created for both the atom and b-factor visualizers to allow the transparency to be adjusted from 0.0 to 1.0.

However, problems with transparency were soon revealed when movies of the atoms (see section 12) started being created. With the transparency parameters that had been chosen, Java could not maintain the depth of the spheres, so they would *pop* in front of each other (Figures 8 & 9). Alternate transparency parameters were selected that did not cause this effect to occur, but they were not nearly as attractive, so transparency was eliminated as an option.

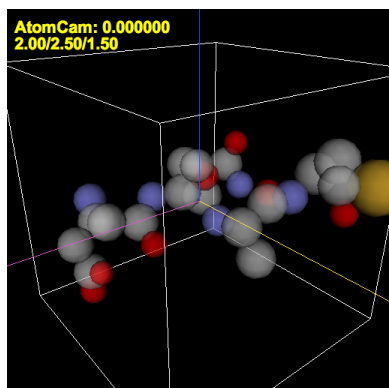


Figure 8: A frame with transparency

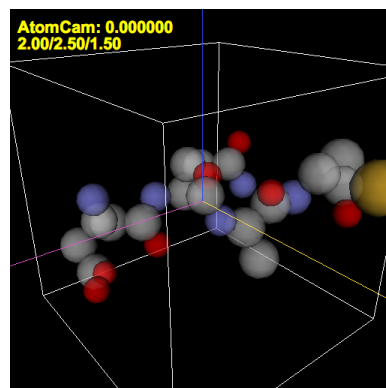


Figure 9: Same frame re-visualized

## 9 Filtering Atoms

After the project had yielded useful images, another requirement emerged. In proteins, there are often a number of atoms around the perimeter that should ideally be removed. In many cases, they are just assumed to be there. By visualizing the entire protein (Figure 10), the resulting image was more complex than it needed to be. This suggested a need for a mechanism to filter atoms out of the protein as it went through the pipeline. This in turn necessitated a way to feed strings into the pipeline, so a `ConstantString` block was added.

The filter works by simply creating a new `AtomIdMap` and only adding `PdbAtoms` to it when they passed a criteria. To process the criteria, a simple expression parser was written. A future improvement may be to move to Java Expression Parser (JEP) because it is already used in Spiegel and can process more complex expressions.

Filters were created that filter based on the name, e.g. Carbon (C), and residue name, e.g. Water (HOH) (Figure 11). In the case of atom name, the filter is a *startsWith* filter because the atom name can have additional information after it, e.g. Alpha-Carbon is CA. The residue filter is a straightforward equality check. A Spiegel user function was also created called *NoWaters* (Figure 12), which filters out the waters from a data path without the need of any input.

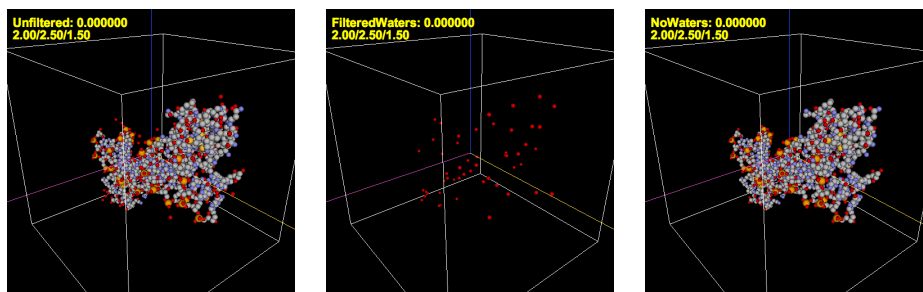


Figure 10: Unfiltered Protein      Figure 11: Filtered Waters      Figure 12: Filtered Protein

## 10 A Specialized Atom Camera

In section 5, a light was added to the scene to make the atoms appear as proper spheres. However, that light was added at a fixed location, so when the camera changed position, the light didn't. This had the effect that at certain camera positions, the atoms would be poorly lit or possibly completely dark. The solution to this was to create an *AtomLight* block that accepts a position parameter.

Instead of giving the camera and the *AtomLight* both a position and adding even more connections to the window, an *AtomCam* user function was added that encapsulates these connections (Figure 13). Now, the light always shines from the camera location at the atom, so the atoms are always illuminated fully. A 3D cube and a set of axes were also added to orient the image a bit better.

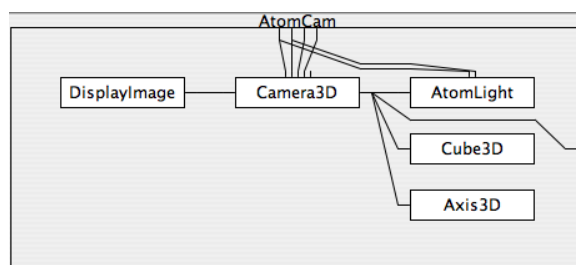


Figure 13: The blocks making up the *AtomCam*

## 11 Intersections

The main goal of the project was to analyze atom intersections within the protein to determine if multiple atoms were occupying the same space. With all of the groundwork of the previous sections laid, it became possible to start looking at the intersections. As an initial implementation, existing Java classes and methods were used to determine intersections. The `BoundingSphere` class offers an `intersect` method that can be used to determine if two `BoundingSpheres` intersect. This was combined with a simple algorithm that compares an atom with all of the remaining atoms. This algorithm has the undesirable  $O(n!)$  complexity, which would be an obvious point of improvement in the future. A method that has been suggested would be to partition the space into boxes and only compare atoms with each box with each other.

Where intersections were detected, a point was rendered at the midpoint between the two atoms. These points were rendered as-is and also with a line connecting them, however, this second method proved not to provide any additional information.

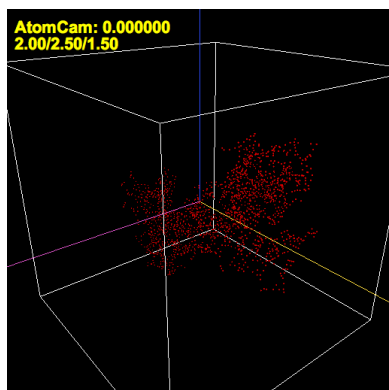


Figure 14: Atom intersections  
(~1700)

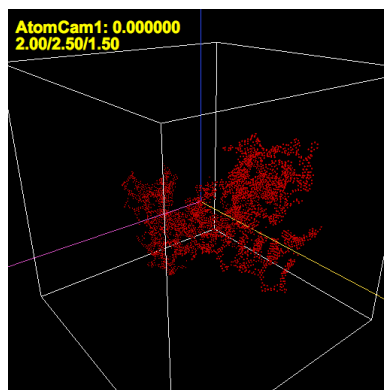


Figure 15: B-Factor intersections  
(~4800)

The assumption when beginning this project was that an initial visualization would not show any intersections; that only when the temperature was increased would intersections start to appear. It was immediately obvious that this was not the case. For the example protein (1D66, ~1700 atoms), approximately 1700 intersections were detected (Figure 14), *without the b-factors*. When b-factors were considered, the number of intersections jumped to approximately 4800 (Figure 15).

Upon consultation with the domain expert, Paul Craig, it was determined that this could possibly be occurring because of the tendency of atoms to shift positions when bonded and sharing electrons. As of the writing of this paper, no mathematical model to counteract this behavior has been found.

## 12 Making Movies

The final task for this project was to make a number of movies to demonstrate the results of the project and the extensibility of Spiegel. This was accomplished by instructing Spiegel to generate files from the images being visualized and then moving the camera along a prescribed path. This path was generated by using the flythrough system implemented in [fly]. Because the flythrough system was developed for visualizing galaxies, it was a rough fit for proteins and could not be used directly. In the future, it is hoped that this system could be used to directly generate movies. There is also another system under development in Spiegel, the Feeder, which may be used to generate movies. The generated movies may be viewed on the author's website [sco].

## 13 Conclusion

The main goal of the project was to analyze atom intersections within the protein to determine if multiple atoms were occupying the same space. In this aspect, the project was unsuccessful as the visualizations generated were not adequate enough to determine true intersections. The project had a secondary goal of demonstrating the extensibility of Spiegel, which was an unequivocal success. Spiegel was shown to be a very capable visualization platform that provides all the tools necessary visualize disparate domains. A few shortcomings of Spiegel were also revealed, yielding a number of possible future improvement projects.

## 14 Future Work

Throughout this paper, a number of possible future improvements on existing techniques have been suggested. These improvements can be made, but the fundamental problem discovered in section 11 of atomic bonds and intersections must be solved before significant forward progress can be made.

## References

- [BWF<sup>+</sup>00] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank, 2000.
- [cpk] Rasmol v2.6 manual.  
<http://info.bio.cmu.edu/Courses/BiochemMols/RasFrames/CPKCLRS.HTM>.
- [fly] A flythrough system for spiegel.  
[http://www.cs.rit.edu/~grapecluster/whitepapers/tim\\_peterson\\_20043.pdf](http://www.cs.rit.edu/~grapecluster/whitepapers/tim_peterson_20043.pdf).
- [gra] The grape cluster project. <http://www.cs.rit.edu/~grapecluster/>.
- [Jmo] The jmol home page. <http://jmol.sourceforge.net/>.
- [Rho00] Gale Rhodes. *Crystallography Made Crystal Clear, 2nd ed., (2000)*. Academic Press, 2000.
- [sco] The website of edward dale.  
[http://www.scompt.com/school/classes/independent\\_study](http://www.scompt.com/school/classes/independent_study).
- [siz] environmentalchemistry.com.  
<http://environmentalchemistry.com/yogi/periodic/atomicradius.html>.