Modeling and visualizing three dimensional crystal structures

Michael Bostick
Mentor: Prof. Stanton
Department of Physics, University of Florida
July 30, 2003

Abstract

This project was designed to achieve understanding of the theory and practice of modeling for solid state physics. The main project focus was on dimensional points of solid crystals. Utilizing Open Data Explorer (OpenDX) it was theoretically possible to visually render three dimensional (3D) points of crystals which were previously to be generated from known mathematical vectors. However, generating data for a crystal was found to be only possible in a programming language such as C plus plus (C++), as the computation of variables is easily done per point rather than per a set of vectors in OpenDX. If all were successful the next step would have been to render these crystal data structures, with attaching bonds, in 3D viewable space as well implementing a virtual space using 3D stereovision. However, much of the project could not be completed as many compiling, rendering, and computational errors arose.

Introduction

Through my project, I am emerged in elementary understandings and conventions of solid state physics. However, making more use of my background knowledge, I began my project programming in Open Data Explorer, (OpenDX). OpenDX is a visual programming language that will generate visual representations of a wide range of data types in a wide range of ways. While this programming language has many complex and modest aspirations, my favorite program in OpenDX is an included example program. This program rendered elevation data for the State of Florida, some surrounding states, and the under belly of the Gulf of Mexico realistically in three dimensions (3D). Many sample programs like this one enabled me to grasp some concepts I would later use such as coloring, computing, and 3D rendering of data. My programming was also to include many modes to view inputted crystal data, but one that was hoped to be implemented is referred to as 3D stereovision. Stereovision is a technique generally used to view flat print in visual virtual 3D without the expense of holograms. Utilizing two distinct images to be viewed by separate eyes, these stereo image pairs come to life by 'refocusing' the human eye into perceiving virtual 3D perspective. It is my hope that I will be able to render crystal structures correctly in OpenDX and if possible using 3D stereovision as well.

Concentrating on crystal structure and replication, I could program in OpenDX to some extent simple, body, and face centered crystals. With selectable exponential crystal sizes, the program was mainly limited by computer memory, as a 30x30x30 fully rotate-able cube becomes cumbersome on the computer's computational powers. However, the crystals I rendered in OpenDX were too simplistic to justify stereovision as a powerful visual aid. From an introductory book on solid state physics, I learned that more advanced crystals behave on principles similar to the simpler ones I modeled, but I was missing dynamic modeling. Instead of a preprogrammed crystal generation, it boils down to knowing a crystal's primitive and basis vectors to define shape, complexity, and size of patterns. The primitive vectors will mainly define structure while the basis vectors place a multiple of atoms at each structure point. The sum of these vectors will in theory produce all the necessary data to visualize in OpenDX. Generating this data, however, could not be accomplished in OpenDx alone, so I wrote an "input vectors, output data points" program in C

plus plus (C++). C++ programming language allows me to perform more drawn out computations per point before saving each as data. This data would in essence be points related to each other by the same 3D space to be rendered by OpenDX. However, some consideration has to be given to efficiency of the model, such as color differentiating between atomic elements in a given crystal. As for separating elements of any given crystal, my options were possibly making data sets for each element or finding a way for OpenDX to differentiate between one point and another.

Hopefully, my project will leave at least some impact in modeling solid crystals through OpenDX. Although the fundamentals of crystal structure are known to an extent beyond the scope of my project, the modeling of such crystals can lead to grander conceptions. Such concepts begin from explaining the free electron of metallic conductivity and can lead to the understanding of amorphous and/or impure materials' interactions and properties. So, as the fundamental crystal structure can be considered a stepping-stone in understanding solid states of matter, my project is a stepping-stone into the process of fundamental modeling of crystals. Studying which modeling theories have failed or succeeded will eventually help in the process of finding the most efficient model to hopefully be applied to the next level of understanding to be modeled as well.

1. OpenDX: methods of visualizations

OpenDX is a visual programming language developed by IBM and was originally licensed under Data Explorer (DX) but is currently open licensed. In addition to generating visualizations for nearly any given input, OpenDX is also easier than most programming languages as it is generally scripted with visual pictograms as the code. These pictograms are known as modules. When these modules are connected to each other, depending on input and output tabs, they form programs that can either export or visualize data [2]. The majority of programming in OpenDX primarily relies on utilizing the numerous standardized modules. However, since OpenDX is open source, many are welcome to create their own modules and even core executables to suit their personal needs or share with other users [5].

I use the modules Glyph and/or AutoGlyph to visualize the coordinate points of each atom in a given crystal [2]. Connecting the data as an entire field of points to glyph will automatically result in rendering each point in 3D space. However, as OpenDX sometimes demands more information rather than tailoring to just the data, other modules are required for a glyph to be displayed. Color and/or AutoColor modules coat the data with rest of the information needed to be displayed. Otherwise OpenDX would complain that you can't see your own data without a color assinged. The color will be important later as it can be set to be dependant on the data hopefully differentiating between atoms when necessary. One of the more difficult parts of sending the data straight to glyph is connecting these atom points with bonds. The bonds are just as important as displaying the atom coordinates, because they are a visual reminder of structure and even the nature/classification of a crystal. The simplest modules to display structure bonds are Construct combined with Show_Connections. These modules can easily be implemented to make cuboids with "data" points at each vertex. See Fig. 1. On the other hand, it is more difficult to dynamically create connections as would Construct require a vector for each line. There would be a need to differentiate between normal bonds and incorrect bonds and at the moment the data is inputted just as structure points. A solution for OpenDX to interpret the proper point connections into vectors is still being investigated.

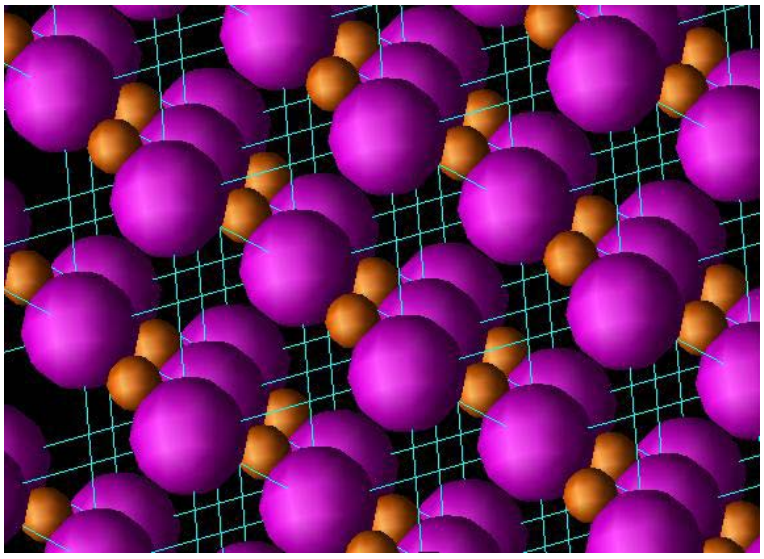

FIG. 1. An OpenDX generated image of a body centered cubic, using default Construct.

The latest test was to determine if the glyphed points could normally translate to Show_Connections as structure vectors and form connecting lines themselves. However, the Show_Connections by itself will only map connections by tracing the order of atoms being displayed. Tracing the atoms by order of display turns out to not only miss most bonds, but also renders connections that are not expected for a given crystal. Another approach would be to create another data file that contains vectors of all the bonds for a given crystal to be displayed in conjunction with its given set of points.

Static data programs such as the one depicted in Fig. 1 require a few pages of programming. Although it can generate exponentially growing body-centered crystal this does not quite fit my project's needs with it's over simplicity. It is more advantageous for the data to originate outside of OpenDX, as in the program in Fig. 2. This program, listed in text format in appendix A, will actually accept generated structure points unlike its predecessor in Fig. 1. However, dynamic as it may be for data points the connection of bonds and coloration are yet to be implemented completely.

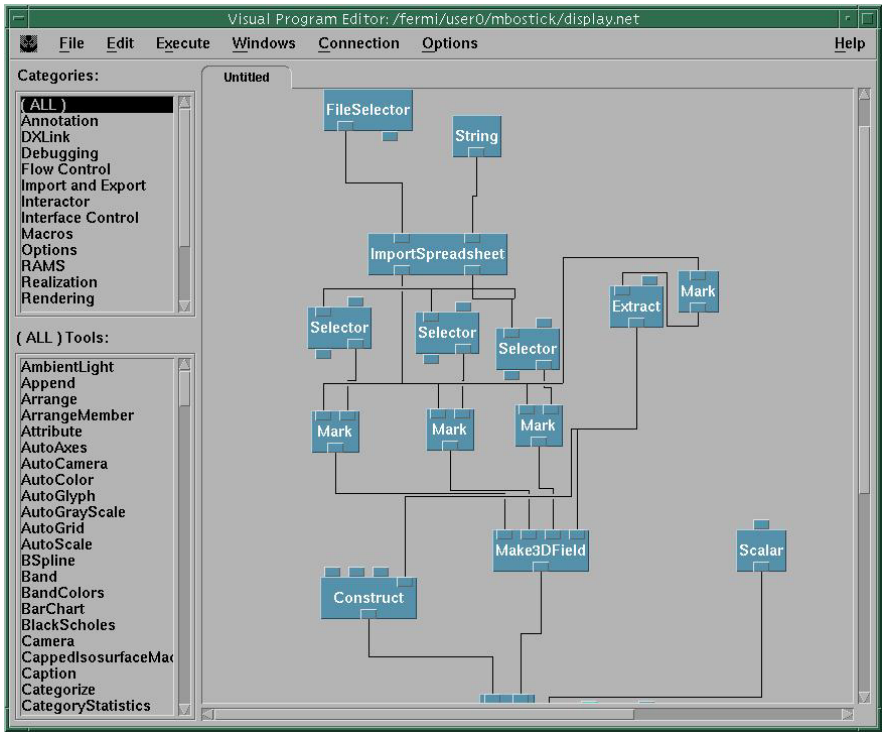

FIG. 2. One page OpenDX program designed to read and display any list of x, y, z sets of data.

The deciding factor in using OpenDX was the possibility of rendering 3D images in stereo 3D. The stereo 3D module had been designed a while ago and had been remade to work with a more recently accepted Open Graphics Language (OpenGL). Although stereovision relies mostly on displaying two camera perspectives at an angle offset to each other, the display of these images simultaneously isn't as simple. Stereovision also relies an additional interface to the monitor. My lab uses CrystalEyes 3D Stereo goggles to control the display of which image is displayed to which eye. With proper timing these goggles will keep one image per eye with electric shutters creating the virtual perception of visual 3D. This would be a significant addition to my project if the complex crystals soon to be generated could be viewed from any angle in display free space and in visual free space. However, despite months of work to install the StereoVision module it will not link properly with the OpenDX executable. I've been in contact with the designer of the recompiled module, but as it was made originally for DX on IBM Sun stations we have not reconciled the problem. Rewriting the module itself might have been just as

successful as solving the linking error; however, at present neither steps can be pursued at great lengths.

2.1 Crystal modeling using Primitive and Basis Vectors

As it turns out, for the level of crystals I am working on, two types of vectors can be used to determine the structure and growth of an entire basic crystal [1]. As a side note, structure points are normally called lattice points and first lattice structures are normally referred to as primitive cells. The main issues, however, are that primitive and basis vectors are enough information to generate simple crystals. Basis vectors for the most part describe positions of atoms depending if the crystal is composed of multiple elements. Another function of basis vectors is to keep track of atoms with a new pattern, as seen in close-packed crystals. The primitive vectors mainly define the structure. Primitive vectors determine skews, heights, patterns of replications, and structure orientations within a crystal.

The current crystal model assumes the summation of each vector with the other will generate all possible points for that vector defined structure. The program written in C++ utilizing this model is attached as appendix B. Initially I thought that starting a cube on a zero plane would help replication and therefore structure, but with summation starting at zero wouldn't help results. But the downsides to building from zero were uniform repetition and possible overlaps in data [4]. On the other hand, the main downside to the summation model is that it's harder to find the center of a structure as repetition can occur in x, y, and/or z direction.

2.2 Connecting specific atoms with bonds

The part of most complexity, which has yet to be investigated, is the coloring and connecting of atoms in a crystal structure given the data. Take for instance the closed packed crystal in Fig. 3. The main solution to look into is breaking up the data depending on the chemical/element group; this would solve the color problem easily as the data starts off seperate and color assignment is just one link away in OpenDX.

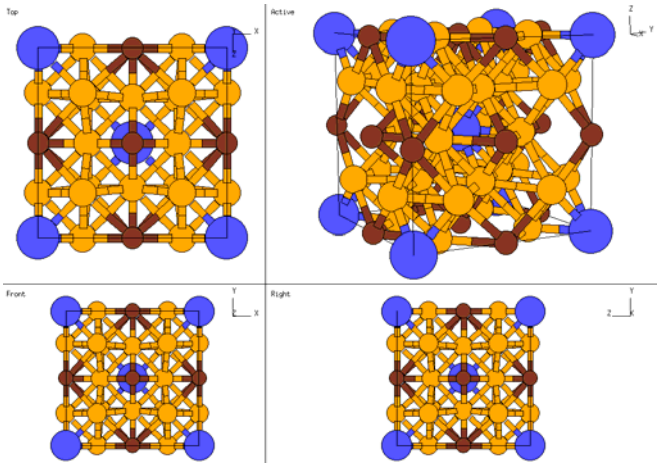

FIG. 3. A Hypothetical cI32 Austenite Structure

*(from **Crystal Lattice Structures** http://cst-www.nrl.navy.mil/lattice/)*

Some structures, however, pose a problem when not every atom has a connection point. This is because such crystals would contain varying symmetry. It appears I still need to find the connection between points in the 3D field, their vectors, and the vectors between each other in order to map out the bonds for a given crystal.

3. Strategy for continuation

As there are as many ways to go about crystal replication, there are just as many problems that I have encountered. It took a while before I realized OpenDX should mainly be used for visualization with as little additional data computation as possible. Computation modules can't directly allow streaming input for computation, but rather these modules compute usually over the entire data sets. Attempting to add a stereovision module proved to be equally as difficult, as the recompiled module was missing a function that would allow it to work with OpenDX. In addition, the crystal data generation was equally as slow due to logical and syntax errors in C++ coding. Although many of these errors are expected to be overcome soon, the entire scope of this project may need further investigation.

Many of the initial problems could have been smoother if there were an expert on OpenDX, module compilation, and C++ input/output idiosyncrasies on standby to help with my

project. However, this was my learning experience using modeling with theory for which I have great appreciation. The next steps for this project include:

- Rendering chemical bonds in OpenDX
- Experimenting with OpenDX data files
- Generating of crystal structure points separated by chemical group/element
- Adding Stereo 3D Vision module to OpenDX or other implementation through OpenDX by OpenGL

Reference

[1] Charles Kittel, *Introduction to Solid State Physics* Seventh Edition

(John Wiley, New York, 1996)

[2] D. Thompson, J. Braun, and R. Ford, *OpenDX Paths to Visualization*

(VIS, Inc, Missoula, 2001)

[3] *Crystal Lattice Structures* (1995) last retrieved July 25, 2003

http://cst-www.nrl.navy.mil/lattice/

[4] *C++ language tutorial* (2002) last retrieved July 25

http://www.cplusplus.com/doc/tutorial/

[5] *Open Visualization Data Explorer* (July 14 2003) last retrieved July 25, 2003

http://www.opendx.org/

Acknowledgments

University of Florida 2003 Physics Research Experience for Undergraduates Program

Special thanks to:

Christopher Stanton – mentor & direction

Tat-Sang Choy – assistance with his Stereo Vision Module

Kevin Ingersent – interest in project

Alan Dorsey – report suggestions

Donna Balkcom – care of paperwork

Brent Nelson & David Hansen – help with program compilation issues

National High Magnetic Field Laboratory in Tallahassee – great lunch and tour

CIRCA CSE E211 – safe use of their computers with faulty segmented code

Very Special thanks to:

Fellow REU Students

Appendix A

```
// display.net
// ################          Text Version of the visual OpenDX program          ##############################
// ################                    minus the comments                       ##############################
// ################                    double blocked to save paper             ##############################
include "Make3DFieldMacro.net"
include "UnsquishGlyphMacro.net"
macro main(
) -> (
) {
main_ImportSpreadsheet_2_out_1,
main_ImportSpreadsheet_2_out_2 =
   ImportSpreadsheet(
   main_FileSelector_2_out_1,
   main_String_2_out_1,
   main_ImportSpreadsheet_2_in_3,
   main_ImportSpreadsheet_2_in_4,
   main_ImportSpreadsheet_2_in_5,
   main_ImportSpreadsheet_2_in_6,
   main_ImportSpreadsheet_2_in_7,
   main_ImportSpreadsheet_2_in_8,
   main_ImportSpreadsheet_2_in_9,
   main_ImportSpreadsheet_2_in_10
   ) [instance: 2, cache: 1];
main_Selector_7_out_1[cache: 2],
main_Selector_7_out_2[cache: 2] =
   Selector(
   main_Selector_7_in_1,
   main_Selector_7_in_2,
   main_Selector_7_in_3,
   main_ImportSpreadsheet_2_out_2,
   main_Selector_7_in_5,
   main_Selector_7_in_6,
   main_Selector_7_in_7
   ) [instance: 7, cache: 1];
main_Mark_6_out_1 =
   Mark(
   main_ImportSpreadsheet_2_out_1,
   main_Selector_7_out_2
   ) [instance: 6, cache: 1];
main_Selector_8_out_1[cache: 2],
main_Selector_8_out_2[cache: 2] =
   Selector(
   main_Selector_8_in_1,
   main_Selector_8_in_2,
   main_Selector_8_in_3,
   main_ImportSpreadsheet_2_out_2,
   main_Selector_8_in_5,
   main_Selector_8_in_6,
   main_Selector_8_in_7
   ) [instance: 8, cache: 1];
main_Mark_7_out_1 =
   Mark(
   main_ImportSpreadsheet_2_out_1,
   main_Selector_8_out_2
   ) [instance: 7, cache: 1];
main_Selector_9_out_1[cache: 2],
main_Selector_9_out_2[cache: 2] =
   Selector(
   main_Selector_9_in_1,
   main_Selector_9_in_2,
   main_Selector_9_in_3,
   main_ImportSpreadsheet_2_out_2,
   main_Selector_9_in_5,
   main_Selector_9_in_6,
   main_Selector_9_in_7
   ) [instance: 9, cache: 1];
main_Mark_8_out_1 =
   Mark(
   main_ImportSpreadsheet_2_out_1,
   main_Selector_9_out_2
   ) [instance: 8, cache: 1];
main_Mark_5_out_1 =
   Mark(
   main_ImportSpreadsheet_2_out_1,
   main_Mark_5_in_2
   ) [instance: 5, cache: 1];
main_Extract_19_out_1 =
   Extract(
   main_Mark_5_out_1,
   main_Extract_19_in_2
   ) [instance: 19, cache: 1];
main_Make3DField_2_out_1 =
   Make3DField(
   main_Mark_6_out_1,
   main_Mark_7_out_1,
   main_Mark_8_out_1,
   main_Extract_19_out_1
   ) [instance: 2, cache: 1];
main_ImportSpreadsheet_1_out_1,
main_ImportSpreadsheet_1_out_2 =
   ImportSpreadsheet(
   main_FileSelector_1_out_1,
   main_String_1_out_1,
   main_ImportSpreadsheet_1_in_3,
   main_ImportSpreadsheet_1_in_4,
   main_ImportSpreadsheet_1_in_5,
   main_ImportSpreadsheet_1_in_6,
   main_ImportSpreadsheet_1_in_7,
   main_ImportSpreadsheet_1_in_8,
   main_ImportSpreadsheet_1_in_9,
   main_ImportSpreadsheet_1_in_10
   ) [instance: 1, cache: 1];
   //
   // node Selector[4]: x = 273, y = 261, inputs = 7, label
= Selector
   // input[1]: defaulting = 0, visible = 0, type = 32, value
= "Selector_4"
   // input[2]: defaulting = 0, visible = 0, type = 32, value
= "column0"
   // input[3]: defaulting = 0, visible = 0, type = 29, value
= 0
   // input[4]: defaulting = 1, visible = 1, type =
16777248, value = {"column0" "column1" "column2" }
   // input[5]: defaulting = 1, visible = 1, type =
16777245, value = {0  1  2  }
   // output[1]: visible = 1, type = 29, value = 0
   // output[2]: visible = 1, type = 32, value = "column0"
   //
main_Selector_4_out_1[cache: 2],
main_Selector_4_out_2[cache: 2] =
   Selector(
   main_Selector_4_in_1,
   main_Selector_4_in_2,
   main_Selector_4_in_3,
   main_ImportSpreadsheet_1_out_2,
   main_Selector_4_in_5,
   main_Selector_4_in_6,
   main_Selector_4_in_7
```

```
    ) [instance: 4, cache: 1];
main_Mark_1_out_1 =
    Mark(
    main_ImportSpreadsheet_1_out_1,
    main_Selector_4_out_2
    ) [instance: 1, cache: 1];
main_Selector_5_out_1[cache: 2],
main_Selector_5_out_2[cache: 2] =
    Selector(
    main_Selector_5_in_1,
    main_Selector_5_in_2,
    main_Selector_5_in_3,
    main_ImportSpreadsheet_1_out_2,
    main_Selector_5_in_5,
    main_Selector_5_in_6,
    main_Selector_5_in_7
    ) [instance: 5, cache: 1];
main_Mark_2_out_1 =
    Mark(
    main_ImportSpreadsheet_1_out_1,
    main_Selector_5_out_2
    ) [instance: 2, cache: 1];
main_Selector_6_out_1[cache: 2],
main_Selector_6_out_2[cache: 2] =
    Selector(
    main_Selector_6_in_1,
    main_Selector_6_in_2,
    main_Selector_6_in_3,
    main_ImportSpreadsheet_1_out_2,
    main_Selector_6_in_5,
    main_Selector_6_in_6,
    main_Selector_6_in_7
    ) [instance: 6, cache: 1];
main_Mark_3_out_1 =
    Mark(
    main_ImportSpreadsheet_1_out_1,
    main_Selector_6_out_2
    ) [instance: 3, cache: 1];
main_Mark_4_out_1 =
    Mark(
    main_ImportSpreadsheet_1_out_1,
    main_Mark_4_in_2
    ) [instance: 4, cache: 1];
main_Extract_1_out_1 =
    Extract(
    main_Mark_4_out_1,
    main_Extract_1_in_2
    ) [instance: 1, cache: 1];
main_Make3DField_1_out_1 =
    Make3DField(
    main_Mark_1_out_1,
    main_Mark_2_out_1,
    main_Mark_3_out_1,
    main_Extract_1_out_1
    ) [instance: 1, cache: 1];
main_Collect_2_out_1 =
    Collect(
    main_Make3DField_2_out_1,
    main_Make3DField_1_out_1
    ) [instance: 2, cache: 1];
main_AutoGlyph_1_out_1 =
    AutoGlyph(
    main_Collect_2_out_1,
    main_AutoGlyph_1_in_2,
    main_AutoGlyph_1_in_3,
    main_Scalar_1_out_1,
    main_AutoGlyph_1_in_5,
    main_AutoGlyph_1_in_6,
    main_AutoGlyph_1_in_7
    ) [instance: 1, cache: 1];
main_Camera_1_out_1 =
    Camera(
    main_Camera_1_in_1,
    main_Camera_1_in_2,
    main_Camera_1_in_3,
    main_Camera_1_in_4,
    main_Camera_1_in_5,
    main_Camera_1_in_6,
    main_Camera_1_in_7,
    main_Camera_1_in_8,
    main_Camera_1_in_9
    ) [instance: 1, cache: 1];
main_ShowConnections_1_out_1 =
    ShowConnections(
    main_Collect_2_out_1
    ) [instance: 1, cache: 1];
main_Color_1_out_1 =
    Color(
    main_AutoGlyph_1_out_1,
    main_Color_1_in_2,
    main_Color_1_in_3,
    main_Color_1_in_4,
    main_Color_1_in_5
    ) [instance: 1, cache: 1];
    //
    // node Collect[1]: x = 407, y = 879, inputs = 2, label
= Collect
    //
main_Collect_1_out_1 =
    Collect(
    main_ShowConnections_1_out_1,
    main_Color_1_out_1
    ) [instance: 1, cache: 1];
main_Image_1_out_1,
main_Image_1_out_2,
main_Image_1_out_3 =
    Image(
    main_Image_1_in_1,
    main_Collect_1_out_1,
    main_Image_1_in_3,
    main_Image_1_in_4,
    main_Image_1_in_5,
    main_Image_1_in_6,
    main_Image_1_in_7,
    main_Image_1_in_8,
    main_Image_1_in_9,
    main_Image_1_in_10,
    main_Image_1_in_11,
    main_Image_1_in_12,
    main_Image_1_in_13,
    main_Image_1_in_14,
    main_Image_1_in_15,
    main_Image_1_in_16,
    main_Image_1_in_17,
    main_Image_1_in_18,
    main_Image_1_in_19,
    main_Camera_1_out_1,
    main_Reset_2_out_1,
    main_Image_1_in_22,
    main_Image_1_in_23,
    main_Image_1_in_24,
    main_Image_1_in_25,
    main_Image_1_in_26,
    main_Image_1_in_27,
    main_Image_1_in_28,
    main_Image_1_in_29,
    main_Image_1_in_30,
    main_Image_1_in_31,
    main_Image_1_in_32,
    main_Image_1_in_33,
```

```
    main_Image_1_in_34,
    main_Image_1_in_35,
    main_Image_1_in_36,
    main_Image_1_in_37,
    main_Image_1_in_38,
    main_Image_1_in_39,
    main_Image_1_in_40,
    main_Image_1_in_41,
    main_Image_1_in_42,
    main_Image_1_in_43,
    main_Image_1_in_44,
    main_Image_1_in_45,
    main_Image_1_in_46,
    main_Image_1_in_47,
    main_Image_1_in_48,
    main_Image_1_in_49
    ) [instance: 1, cache: 1];

main_UnsquishGlyph_1_out_1 =
    UnsquishGlyph(
    main_UnsquishGlyph_1_in_1,
    main_UnsquishGlyph_1_in_2
    ) [instance: 1, cache: 1];
// network: end of macro body
CacheScene(main_Image_1_in_1,
main_Image_1_out_1, main_Image_1_out_2);
}
main_FileSelector_2_out_1 =
"/fermi/user0/mbostick/bccP.txt";
main_String_2_out_1 = " ";
main_ImportSpreadsheet_2_in_3 = NULL;
main_ImportSpreadsheet_2_in_4 = NULL;
main_ImportSpreadsheet_2_in_5 = NULL;
main_ImportSpreadsheet_2_in_6 = NULL;
main_ImportSpreadsheet_2_in_7 = NULL;
main_ImportSpreadsheet_2_in_8 = NULL;
main_ImportSpreadsheet_2_in_9 = NULL;
main_ImportSpreadsheet_2_in_10 = NULL;
main_ImportSpreadsheet_2_out_1 = NULL;
main_ImportSpreadsheet_2_out_2 = NULL;
main_Selector_7_in_1 = "Selector_7";
main_Selector_7_in_2 = "column0" ;
main_Selector_7_in_3 = 0 ;
main_Selector_7_in_5 = NULL;
main_Selector_7_in_6 = NULL;
main_Selector_7_in_7 = NULL;
main_Selector_7_out_2 = "column0" ;
main_Mark_6_out_1 = NULL;
main_Selector_8_in_1 = "Selector_8";
main_Selector_8_in_2 = "column1" ;
main_Selector_8_in_3 = 1 ;
main_Selector_8_in_5 = NULL;
main_Selector_8_in_6 = NULL;
main_Selector_8_in_7 = NULL;
main_Selector_8_out_2 = "column1" ;
main_Mark_7_out_1 = NULL;
main_Selector_9_in_1 = "Selector_9";
main_Selector_9_in_2 = "column2" ;
main_Selector_9_in_3 = 2 ;
main_Selector_9_in_5 = NULL;
main_Selector_9_in_6 = NULL;
main_Selector_9_in_7 = NULL;
main_Selector_9_out_2 = "column2" ;
main_Mark_8_out_1 = NULL;
main_Mark_5_in_2 = "column0";
main_Mark_5_out_1 = NULL;
main_Extract_19_in_2 = "data";
main_Extract_19_out_1 = NULL;
main_Make3DField_2_out_1 = NULL;
main_FileSelector_1_out_1 =
"/fermi/user0/mbostick/Primitive.txt";
main_String_1_out_1 = " ";
main_ImportSpreadsheet_1_in_3 = NULL;
main_ImportSpreadsheet_1_in_4 = NULL;
main_ImportSpreadsheet_1_in_5 = NULL;
main_ImportSpreadsheet_1_in_6 = NULL;
main_ImportSpreadsheet_1_in_7 = NULL;
main_ImportSpreadsheet_1_in_8 = NULL;
main_ImportSpreadsheet_1_in_9 = NULL;
main_ImportSpreadsheet_1_in_10 = NULL;
main_ImportSpreadsheet_1_out_1 = NULL;
main_ImportSpreadsheet_1_out_2 = NULL;
main_Selector_4_in_1 = "Selector_4";
main_Selector_4_in_2 = "column0" ;
main_Selector_4_in_3 = 0 ;
main_Selector_4_in_5 = NULL;
main_Selector_4_in_6 = NULL;
main_Selector_4_in_7 = NULL;
main_Selector_4_out_2 = "column0" ;
main_Mark_1_out_1 = NULL;
main_Selector_5_in_1 = "Selector_5";
main_Selector_5_in_2 = "column1" ;
main_Selector_5_in_3 = 1 ;
main_Selector_5_in_5 = NULL;
main_Selector_5_in_6 = NULL;
main_Selector_5_in_7 = NULL;
main_Selector_5_out_2 = "column1" ;
main_Mark_2_out_1 = NULL;
main_Selector_6_in_1 = "Selector_6";
main_Selector_6_in_2 = "column2" ;
main_Selector_6_in_3 = 2 ;
main_Selector_6_in_5 = NULL;
main_Selector_6_in_6 = NULL;
main_Selector_6_in_7 = NULL;
main_Selector_6_out_2 = "column2" ;
main_Mark_3_out_1 = NULL;
main_Mark_4_in_2 = "column0";
main_Mark_4_out_1 = NULL;
main_Extract_1_in_2 = "data";
main_Extract_1_out_1 = NULL;
main_Make3DField_1_out_1 = NULL;
main_Collect_2_out_1 = NULL;
main_Scalar_1_in_1 = "Scalar_1";
main_Scalar_1_in_2 = NULL;
main_Scalar_1_in_3 = 0.65 ;
main_Scalar_1_in_4 = NULL;
main_Scalar_1_in_5 = NULL;
main_Scalar_1_in_6 = NULL;
main_Scalar_1_in_7 = NULL;
main_Scalar_1_in_8 = NULL;
main_Scalar_1_in_9 = NULL;
main_Scalar_1_in_10 = NULL;
main_Scalar_1_in_11 = NULL;
main_Scalar_1_out_1 = 0.65 ;
main_AutoGlyph_1_in_2 = "spiffy";
main_AutoGlyph_1_in_3 = NULL;
main_AutoGlyph_1_in_5 = 1.0;
main_AutoGlyph_1_in_6 = NULL;
main_AutoGlyph_1_in_7 = NULL;
main_AutoGlyph_1_out_1 = NULL;
main_Camera_1_in_1 = [9 9 9];
main_Camera_1_in_2 = [19 19 19];
main_Camera_1_in_3 = NULL;
main_Camera_1_in_4 = NULL;
main_Camera_1_in_5 = 1.0;
main_Camera_1_in_6 = [0 1 0];
main_Camera_1_in_7 = 1;
main_Camera_1_in_8 = 0.0;
main_Camera_1_in_9 = NULL;
```

```
main_Camera_1_out_1 = NULL;
main_ShowConnections_1_out_1 = NULL;
main_Color_1_in_2 = "orange";
main_Color_1_in_3 = NULL;
main_Color_1_in_4 = NULL;
main_Color_1_in_5 = NULL;
main_Color_1_out_1 = NULL;
main_Collect_1_out_1 = NULL;
main_Reset_2_in_1 = "main_Reset_2_out_1";
main_Reset_2_in_2 = 0;
main_Reset_2_in_3 = 0;
main_Reset_2_in_4 = NULL;
main_Reset_2_in_5 = NULL;
main_Reset_2_in_6 = NULL;
main_Reset_2_out_1 = 0;
macro Image(
        id,
        object,
        where,
        useVector,
        to,
        from,
        width,
        resolution,
        aspect,
        up,
        viewAngle,
        perspective,
        options,
        buttonState = 1,
        buttonUpApprox = "none",
        buttonDownApprox = "none",
        buttonUpDensity = 1,
        buttonDownDensity = 1,
        renderMode = 0,
        defaultCamera,
        reset,
        backgroundColor,
        throttle,
        RECenable = 0,
        RECfile,
        RECformat,
        RECresolution,
        RECaspect,
        AAenable = 0,
        AAlabels,
        AAticks,
        AAcorners,
        AAframe,
        AAadjust,
        AAcursor,
        AAgrid,
        AAcolors,
        AAannotation,
        AAlabelscale,
        AAfont,
        interactionMode,
        title,
        AAxTickLocs,
        AAyTickLocs,
        AAzTickLocs,
        AAxTickLabels,
        AAyTickLabels,
        AAzTickLabels,
        webOptions) -> (
        object,
        camera,
        where)
{
    ImageMessage(
        id,
        backgroundColor,
        throttle,
        RECenable,
        RECfile,
        RECformat,
        RECresolution,
        RECaspect,
        AAenable,
        AAlabels,
        AAticks,
        AAcorners,
        AAframe,
        AAadjust,
        AAcursor,
        AAgrid,
        AAcolors,
        AAannotation,
        AAlabelscale,
        AAfont,
        AAxTickLocs,
        AAyTickLocs,
        AAzTickLocs,
        AAxTickLabels,
        AAyTickLabels,
        AAzTickLabels,
        interactionMode,
        title,
        renderMode,
        buttonUpApprox,
        buttonDownApprox,
        buttonUpDensity,
        buttonDownDensity) [instance: 1, cache: 1];
    autoCamera =
        AutoCamera(
            object,
            "front",
            object,
            resolution,
            aspect,
            [0,1,0],
            perspective,
            viewAngle,
            backgroundColor) [instance: 1, cache: 1];
    realCamera =
        Camera(
            to,
            from,
            width,
            resolution,
            aspect,
            up,
            perspective,
            viewAngle,
            backgroundColor) [instance: 1, cache: 1];
    coloredDefaultCamera =
                UpdateCamera(defaultCamera,
            background=backgroundColor) [instance: 1,
cache: 1];
    nullDefaultCamera =
        Inquire(defaultCamera,
            "is null + 1") [instance: 1, cache: 1];
    resetCamera =
        Switch(
            nullDefaultCamera,
            coloredDefaultCamera,
            autoCamera) [instance: 1, cache: 1];
    resetNull =
        Inquire(
            reset,
```

```
        "is null + 1") [instance: 2, cache: 1];
    reset =
        Switch(
            resetNull,
            reset,
            0) [instance: 2, cache: 1];
    whichCamera =
        Compute(
            "($0 != 0 || $1 == 0) ? 1 : 2",
            reset,
            useVector) [instance: 1, cache: 1];
    camera = Switch(
            whichCamera,
            resetCamera,
            realCamera) [instance: 3, cache: 1];
    AAobject =
        AutoAxes(
            object,
            camera,
            AAlabels,
            AAticks,
            AAcorners,
            AAframe,
            AAadjust,
            AAcursor,
            AAgrid,
            AAcolors,
            AAannotation,
            AAlabelscale,
            AAfont,
            AAxTickLocs,
            AAyTickLocs,
            AAzTickLocs,
            AAxTickLabels,
            AAyTickLabels,
            AAzTickLabels) [instance: 1, cache: 1];
    switchAAenable = Compute("$0+1",
                AAenable) [instance: 2, cache: 1];
    object = Switch(
                switchAAenable,
                object,
                AAobject) [instance:4, cache: 1];
    SWapproximation_options =
        Switch(
            buttonState,
            buttonUpApprox,
            buttonDownApprox) [instance: 5, cache: 1];
    SWdensity_options =
        Switch(
            buttonState,
            buttonUpDensity,
            buttonDownDensity) [instance: 6, cache: 1];
    HWapproximation_options =
        Format(
            "%s,%s",
            buttonDownApprox,
            buttonUpApprox) [instance: 1, cache: 1];
    HWdensity_options =
        Format(
            "%d,%d",
            buttonDownDensity,
            buttonUpDensity) [instance: 2, cache: 1];
    switchRenderMode = Compute(
                "$0+1",
                renderMode) [instance: 3, cache: 1];
    approximation_options = Switch(
                switchRenderMode,
            SWapproximation_options,
                HWapproximation_options) [instance: 7,
cache: 1];
    density_options = Switch(
                switchRenderMode,
            SWdensity_options,
            HWdensity_options) [instance: 8, cache: 1];
    renderModeString = Switch(
            switchRenderMode,
            "software",
            "hardware")[instance: 9, cache: 1];
    object_tag = Inquire(
            object,
            "object tag")[instance: 3, cache: 1];
    annotated_object =
        Options(
            object,
            "send boxes",
            0,
            "cache",
            1,
            "object tag",
            object_tag,
            "ddcamera",
            whichCamera,
            "rendering approximation",
            approximation_options,
            "render every",
            density_options,
            "button state",
            buttonState,
            "rendering mode",
            renderModeString) [instance: 1, cache: 1];
    RECresNull =
        Inquire(
            RECresolution,
            "is null + 1") [instance: 4, cache: 1];
    ImageResolution =
        Inquire(
            camera,
            "camera resolution") [instance: 5, cache: 1];
    RECresolution =
        Switch(
            RECresNull,
            RECresolution,
            ImageResolution) [instance: 10, cache: 1];
    RECaspectNull =
        Inquire(
            RECaspect,
            "is null + 1") [instance: 6, cache: 1];
    ImageAspect =
        Inquire(
            camera,
            "camera aspect") [instance: 7, cache: 1];
    RECaspect =
        Switch(
            RECaspectNull,
            RECaspect,
            ImageAspect) [instance: 11, cache: 1];
    switchRECenable = Compute(
          "$0 == 0 ? 1 : (($2 == $3) && ($4 == $5)) ? ($1
== 1 ? 2 : 3) : 4",
            RECenable,
            switchRenderMode,
            RECresolution,
            ImageResolution,
            RECaspect,
                ImageAspect) [instance: 4, cache: 1];
    NoRECobject, RECNoRerenderObject,
RECNoRerHW, RECRerenderObject =
Route(switchRECenable, annotated_object);
    Display(
        NoRECobject,
```

```
    camera,
    where,
    throttle) [instance: 1, cache: 1];
image =
    Render(
        RECNoRerenderObject,
        camera) [instance: 1, cache: 1];
Display(
    image,
    NULL,
    where,
    throttle) [instance: 2, cache: 1];
WriteImage(
    image,
    RECfile,
    RECformat) [instance: 1, cache: 1];
rec_where = Display(
    RECNoRerHW,
    camera,
    where,
    throttle) [instance: 1, cache: 0];
rec_image = ReadImageWindow(
    rec_where) [instance: 1, cache: 1];
WriteImage(
    rec_image,
    RECfile,
    RECformat) [instance: 1, cache: 1];
RECupdateCamera =
        UpdateCamera(
            camera,
            resolution=RECresolution,
            aspect=RECaspect) [instance: 2, cache:
1];
Display(
    RECRerenderObject,
    camera,
    where,
    throttle) [instance: 1, cache: 1];
RECRerenderObject =
        ScaleScreen(
            RECRerenderObject,
            NULL,
            RECresolution,
            camera) [instance: 1, cache: 1];
image =
    Render(
        RECRerenderObject,
        RECupdateCamera) [instance: 2, cache: 1];
WriteImage(
    image,
    RECfile,
    RECformat) [instance: 2, cache: 1];
}
main_Image_1_in_1 = "Image_1";
main_Image_1_in_3 = "X24,,";
main_Image_1_in_4 = 1;
main_Image_1_in_5 = [6.15482 4.7897 6.12233];
main_Image_1_in_6 = [-6.35116 -0.807943 -4.47328];
main_Image_1_in_7 = 41.2194;
main_Image_1_in_8 = 640;
main_Image_1_in_9 = 1.0;
main_Image_1_in_10 = [0.49665 0.373463 -0.783495];
main_Image_1_in_11 = NULL;
main_Image_1_in_12 = 0;
main_Image_1_in_13 = NULL;
main_Image_1_in_14 = 1;
main_Image_1_in_15 = NULL;
main_Image_1_in_16 = NULL;
main_Image_1_in_17 = NULL;
main_Image_1_in_18 = NULL;
main_Image_1_in_19 = 0;
main_Image_1_in_22 = NULL;
main_Image_1_in_23 = NULL;
main_Image_1_in_25 = "bcc.jpg";
main_Image_1_in_26 = "ImageMagick supported
format";
main_Image_1_in_27 = NULL;
main_Image_1_in_28 = NULL;
main_Image_1_in_29 = NULL;
main_Image_1_in_30 = NULL;
main_Image_1_in_31 = NULL;
main_Image_1_in_32 = NULL;
main_Image_1_in_33 = NULL;
main_Image_1_in_34 = NULL;
main_Image_1_in_35 = NULL;
main_Image_1_in_36 = NULL;
main_Image_1_in_37 = NULL;
main_Image_1_in_38 = NULL;
main_Image_1_in_39 = NULL;
main_Image_1_in_40 = NULL;
main_Image_1_in_41 = "rotate";
main_Image_1_in_42 = NULL;
main_Image_1_in_43 = NULL;
main_Image_1_in_44 = NULL;
main_Image_1_in_45 = NULL;
main_Image_1_in_46 = NULL;
main_Image_1_in_47 = NULL;
main_Image_1_in_48 = NULL;
main_Image_1_in_49 = NULL;
main_UnsquishGlyph_1_in_1 = NULL;
main_UnsquishGlyph_1_in_2 = NULL;
Executive("product version 4 2 0");
$sync
main();
```

Appendix B

```
// gen_prim3.cpp
// ################          Main generation of structure data points          ##############################
// ################          currently produces summation modeled data         ##############################
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <string>
#include <vector>
#include <stdlib.h>
#include <stdio.h>
#include <iomanip>

using namespace std;

#define TRUE 1
#define FALSE 0

int main()
{
        // vector will temporarily hold the primitive vectors
        float vector [3][3]= {(0,0,0),(0,0,0),(0,0,0)};

        // transfer is the first to accept float data extracted from reading variables
        float transfer[3];
        int base = 0;
        int count1 = 0;
        int count2 = 0;
        int k = 0;

        // reading variable declarations and initiations
        char chose;
        char buffer[256];
        for (int i=0;i<256;i++) buffer[i] = '\0';
        char num[12]; char chem_ck[12];
        for (int i=0;i<12;i++) num[i]  = chem_ck[i]= '\0';

        // stores chemical names
        string chem;
        // counts how many chemicals
        int chem_count = 0;
/*
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
*/
// READ PRIMITIVE VECTORS

    cout << endl;
    cout << "Primitive Vectors can be inputted as a file in the form of:" << endl;
    cout << "vectors in (x y z) format" << endl;
    cout << "For Example: 0.000 1.000 2.000" << endl;
    cout << endl;
    cout << "Read primitive vectors from file? (Y)es (N)o :";
    cin >> chose;
        //cin.getline( buffer, 100, '\n' );  //Optional Depends on Compiler
    if (chose == 'Y' || chose == 'y')
    {
        // READ FROM FILE for primitive vectors

        ifstream infile;
        string filename;
        cout << "Please Enter a filename: ";
        cin >> filename;
                //cin.get(); //Optional Depends on Compiler
        infile.open (filename.c_str());
```

```
        // if reading input file fails for primitive, end program
        if (infile.fail())
        { cout << '\n' << "Error opening file"; exit (1); }

        // keep reading while there is data
        while (!infile.eof())
        {
                    // read one line at a time in file
                    infile.getline (buffer, 100);
                    int k=0;
                    int j=0;
                    for (int i =0; i<100; i++)
                    {
                          if (buffer[i]=='\0') i=100;
                          else
                          {
                                if (buffer[i] != ' ')
                                {
                                          // as long as a space isn't encountered transfer line to num
                                      num[j] = buffer[i];
                                      num[j+1]='\0';
                                      j++;
                                } else {
                                      if (k<3)
                                      {
                                           // if a space deliminator is encountered, transfer string as a float
                                          transfer[k] = atof(num);
                                          if (buffer[i+1]!='\0' && buffer[i+1] != ' ')
                                                k++;
                                          j=0;
                                      }
                                }
                          }
                                // transfer last number
                          if (k < 3)
                          transfer[k] = atof(num);
                    }

                    // transfer one vector
                    for (k=0; k<3; k++)
                    {
                          vector[count2][count1] = transfer[k];
                          count1++;
                    }

                    // start next vector
                    if ((count1>=3))
                    {
                           count2++;
                           count1 = 0;
                    }

                    // end after third vector is transfered
                    if ((count2>=3))
                    {
                          count1 = count2 = 4;
                          infile.read (buffer,1);
                    }
        }
              infile.close();
}else{
        // READ KEYBOARD for primitive vectors

        // clear any hanging characters
        cin.getline( buffer, 100, '\n' );
        cout << "Input each primitive vector as one line deliminated by spaces" << endl;
        cout << "For Example: 0.000 1.000 2.000" << endl;
        for (int i = 0; i < 3; i++)
        {
                    cout << ":";
```

```
                        // read each vector from keyboard input
                        cin.getline( buffer, 100, '\n' );

                        int k=0;
                        int j=0;
                        for (int i =0; i<100; i++)
                        {
                                if (buffer[i]=='\0') i=100;
                                else
                                {
                                        if (buffer[i] != ' ')
                                        {
                                                num[j] = buffer[i];
                                                num[j+1]='\0';
                                                j++;
                                        } else {
                                                if (k<3)
                                                {
                                                        transfer[k] = atof(num);
                                                        if (buffer[i+1]!='\0' && buffer[i+1] != ' ')
                                                                k++;
                                                        j=0;
                                        }
                                                                        }
                        }
                        if (k < 3)
                                        transfer[k] = atof(num);
                        }
                        for (k=0; k<3; k++)
                {
                vector[count1][count2] = transfer[k];
                        count1++;
                }
                if ((count1>=3))
                {
                        count2++;
                        count1 = 0;
                }
                if ((count2>=3))
                        count1 = count2 = 4;
                num[0] = '\0';
                        }
        }

                cout << endl;
        cout << "-----------------------------------------" << endl;
        cin.clear();
        chose = '\0';
/*
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
*/
        count1 = count2 = 0;
        bool ok;
        int l = 0;

                // B will store each Basis vector, 20 possible
       float B[3][20];
        int m = 0;
                // chem_match keeps track of how many elements per element
        int chem_match[20];
                // intialize Basis vectors to zero
        for (int i=0;i<3;i++)
        {
                for (int j=0;j<20;j++)
                        B[i][j] = 0;
        }

        cout << endl;
```

```
cout << "Basis Vectors can be inputted as a file in the form of:" << endl;
cout << "vectors listed by chemical" << endl;
cout << "in (x y z) & chemical symbol format" << endl;
cout << "For Example: 0.000 1.000 2.000 Fe" << endl;
cout << endl;
cout << "Read basis vectors from file? (Y)es (N)o :";
cin >> chose;
	//cin.getline( buffer, 100, '\n' );  //Optional Depends on Compiler
if (chose == 'Y' || chose == 'y')
{
	// READ BASIS FROM FILE
	ifstream infile;
	string filename;
	cout << "Please Enter a filename: ";
	cin >> filename;
	//cin.get();  //Optional Depends on Compiler
	infile.open (filename.c_str());

	// if Basis file read fails set basis to 0 0 0 NA
	if (infile.fail())
	{
		cout << '\n' << "Error opening file";
		cout << "continuing as if basis was 0 0 0 NA" << endl;
		base = 1;
		for (int i=0;i<3;i++)
			B[i][0] = 0;
		chem = 'N' + 'A';
	}else{
		while (!infile.eof())
		{
			infile.getline (buffer, 100);
			int k=0;
			int j=0;
			for (int i =0; i<100; i++)
			{
				if (buffer[i]=='\0')
				{
					// \0 would indicate an entire line has been read from buffer
					// therefore the chemical name to be extracted must be in num

					i=100; // exit buffer loop after two checks for chem name

					// check if the next chemical is new
					l=0;
					ok = FALSE;
					while (num[l] != '\0')
					{
						if (num[l] != chem_ck[l])
							ok = TRUE;
						l++;
					}

					// if its a new chemical and is under limits add to list
					if (ok  && chem_count<=4 || chem_count==0)
					{
							strcpy(chem_ck, num);
							chem = chem + chem_ck + ' ';

							// keeps track of the number of unique chemical names
							// **requirement all chemicals are listed by name
							chem_match[chem_count-1] = m;
							 chem_count++;
							chem_match[chem_count-1] = m; chem_match[chem_count]='\0';
							m=0;
					}
					k++;
				}else{
					// as long as buffer has not reached its end read buffer into num

					// check for space deliminator
```

```
                        if (buffer[i] != ' ')
                        {
                                num[j] = buffer[i];
                                num[j+1]='\0';
                                j++;
                        } else {
                         // if space deliminator encountered, transfer data point from num
                                if (k<3)
                                {
                                        transfer[k] = atof(num);
                                        if (buffer[i+1]!='\0' && buffer[i+1] != ' ')
                                        k++;
                                        j=0;
                                }
                        }
                      }
                      if (k < 3)
                      transfer[k] = atof(num);
                }

                // transfer xyz coordinates to B(asis) vector
                 for (k=0; k<3; k++)
                {
                        B[count1][count2] = transfer[k];
                        count1++;
                }

                // whenever xyz are all transfered move to next vector row and reset column
                if ((count1>=3))
                {
                         count2++;

                // variable base is similar to m
                // difference: base is how many vectors total
                        base++;
                // m is how many vectors a given element
                        m++;

                         count1 = 0;
                }
                if (!infile)
                {
                // exit all reads and transfers
                        count1 = count2 = 4;
                        infile.read (buffer,1);
                }
        }
        }
                infile.close();
} else {
        // READ BASIS if not to be read from a file

                cout << "How many basis vectors? ";
                        cin >> base;
        cin.get();
        //cin.get();//Optional Depends on Compiler
                if (base >= 1)
                        {
                                cout << "Input each base vector on one line deliminated by spaces" << endl;
                                cout << "Numbers first, chemical symbols last" << endl;
                                cout << "For Example: 0.000 1.000 2.000 Fe" << endl;

                         for (int i = 0; i < base; i++)
                         {
                                cout << ":";
                                cin.getline( buffer, 100, '\n' );
                                int k=0;
                                int j=0;
                        for (int i =0; i<100; i++)
                        {
```

```
                        if (buffer[i]=='\0')
                        {
                                i=100;
                                l=0;
                                ok = FALSE;
                                while (num[l] != '\0')
                                {
                                        if (num[l] != chem_ck[l])
                                                ok = TRUE;
                                        l++;
                                }
                                if (ok  && chem_count<=4 || chem_count==0)
                                {
                                        strcpy(chem_ck, num);
                                        chem = chem + chem_ck + ' ';
                                        chem_match[chem_count-1] = m;
                                         chem_count++;
                                         chem_match[chem_count-1] = m; chem_match[chem_count]='\0';
                                        m=0;
                                }
                                k++;
                        }else{
                                if (buffer[i] != ' ')
                                {
                                        num[j] = buffer[i];
                                        num[j+1]='\0';
                                        j++;
                                } else {
                                        if (k<3)
                                        {
                                                transfer[k] = atof(num);
                                                if (buffer[i+1]!='\0' && buffer[i+1] != ' ')
                                                k++;
                                                j=0;
                                        }
                                }
                        }
                        if (k < 3)
                                transfer[k] = atof(num);
                }
                for (k=0; k<3; k++)
                {
                      B[count1][count2] = transfer[k];
                       count1++;
                }
                if ((count1>=3))
                {
                        count2++;
                        m++;
                        count1 = 0;
                }
                if ((count2>=base))
                        count1 = count2 = 4;
        }
        }else{
                        base = 1;
                        for (int i=0;i<3;i++)
                                B[i][0] = 0;
                        chem = 'N' + 'A';
                        cout << "Basis was set to (0 0 0 Na)" << endl;
                        }
        }
        chem_match[chem_count-1] = m;
/*
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
*/
        cout << "-----------------------------------------" << endl;
// READ IN REPETITIONS
```

```
		int rep[3];
		cin.clear();
		cout << endl;
		cout << "Please enter the Number of Repititions in the following Axis:" << endl;
		cout << "X-Axis: ";
		cin >> rep[0];
		cin.get(); //Unix Compile Only

		cout << "Y-Axis: ";
		cin >> rep[1];
		cin.get(); //Unix Compile Only

		cout << "Z-Axis: ";
		cin >> rep[2];
		cin.get(); //Unix Compile Only


		for (int i = 0; i < 3; i++)
		{
			// make sure its not too large
			// 7x7x7 = 343
			// 343 + 000 + all basis(20) + all primitives(3) = 367 total vectors possible
			// see notes for expansion
			if (rep[i] > 7)
				rep[i] = 7;

			// make sure its not too small
			if (rep[i] < 2)
				rep[i] = 2;
		}


/*
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
*/
		// PRINT INPUT PRIMITIVES
	cout << "-----------------------------------------" << endl;
		cout << endl << "done" << endl;
	cout << setiosflags(ios::left);
	cout << "PRIMITIVE VECTORS" << endl;
	for (int i=0;i<3;i++)
	{
		for (int j=0;j<3;j++)
		{
			cout << setw(13) << vector[j][i];
			if  (j==2) cout << endl;
		}
	}
		// PRINT INPUT BASIS
	cout << endl << "BASIS VECTORS" << endl;
	int n = m = k = 0;

		// for every base vector recorded (row)
	for (int i=0;i<base;i++)
	{
		// through each coordinate x, y, and z
		for (int j=0;j<3;j++)
		{
			// print point with spacing
			cout << setw(13) <<  B[j][i];

			// if last coordinate printed
			if  (j==2)
			{
				// if retracted too far move up
				while (chem[k] == ' ' && k<20)
					k++;
					// test
```

```
                        // cout << endl << chem_match[n] << "$" << n<< "&"<< k << endl;
                          // if amount of chemical names printed is less than total recorded
                          if (m < chem_match[n])
                          {
                                    cout << setw(13);
                                    // if not last chemical or a white space and under name limit
                                while (chem[k]!='\0' && chem[k] != ' ' && k<20)
                                {
                                        // print chemical name letter at a time
                                        cout << chem[k];
                                        k++;
                                }
                                        // add to count of chemicals printed
                                        m++;

                                        // back track chemical name if chemicals printed is less than recorded
                                        k--;
                                        if (m < chem_match[n])
                                        {
                                        while (chem[k] != ' ' && k>=0)
                                                k--;
                          }

                          // make sure loop ends
                           if (m >= chem_match[n])
                                    m++;
                          }else{
                                    // if chemicals printed is greater than or equal to recorded
                                    cout << setw(13);

                                    // skip to next chemical name
                                    while (chem[k]!='\0' && chem[k] != ' ' && k<20)
                                            k++;
                                    while (chem[k] == ' ' && k<20)
                                            k++;

                                // if not last chemical or a white space and under name limit
                                while (chem[k]!='\0' && chem[k] != ' ' && k<20)
                                {
                                        // print chemical name letter at a time
                                        cout << chem[k];
                                        k++;
                                }
                                         // new count of chemicals printed
                                        m=1;
                                        // next amount in a series of a chemical
                                        n++;
                                         // back track chemical name if chemicals printed is less than recorded
                                        k--;
                                        if (m < chem_match[n])
                                         {
                                                    while (chem[k] != ' ' && k>=0)
                                                            k--;
                                        }
                          }
                          cout << endl;
                    }
               }
          }
          cout << resetiosflags(ios::left);
          cout << endl;

/*
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
*/
          cout << "-----------------------------------------" << endl;
                // OUTPUT TO FILE Primitive.txt
                float A1[3],A2[3],A3[3];
```

```
	// primitive seperation for easier reading
	for (int i=0;i<3;i++)
	{
	A1[i] = vector[i][0];
		A2[i] = vector[i][1];
	A3[i] = vector[i][2];
	}

	float tmpdata [3][967]; n=0;

	for (int L = 0; L < rep[0]; L++)
	{	// x vector propogation

	for (int M = 0; M < rep[1]; M++)
	{	// y vector propogation

	for (int N = 0; N < rep[2]; N++)
		{	// z vector propogation

			for (int atoms = 0; atoms < base; atoms++)
		{
						/* ##Primitive##
				L*A1 + M*A2 + N*A3 + B[atoms] */
		// SUMS EACH VECTOR COMPONENTS
			for (m=0;m<3;m++)
			tmpdata[m][n] = L*A1[m] + M*A2[m] + N*A3[m] + B[m][atoms];
			//	cout << tmpdata[m][n];}
						//cout << endl;
						n++;

					// NOT IMPLEMENTED
						/* ##Base##
						L*A1 + B[atoms]
						L*A2 + B[atoms]
						L*A3 + B[atoms]
						for (m=0;m<3;m++)
			tmpdata[m][n] = L*A1[m] + B[m][atoms];
						n++;
						for (m=0;m<3;m++)
			tmpdata[m][n] = L*A2[m] + B[m][atoms];
			n++;
						for (m=0;m<3;m++)
			tmpdata[m][n] = L*A3[m] + B[m][atoms];
			n++; */
					}
		}
		}
	}

	float data [3][367];
	cout << "ok" << endl;
	for (int i=0; i<n; i++)
	{
		data[0][i] = tmpdata[0][i];
		data[1][i] = tmpdata[1][i];
		data[2][i] = tmpdata[2][i];
	}
	cout << "outputting" << endl;
ofstream outfile;
	outfile.open ("Primitive.txt");
	outfile.clear();
	if (outfile.is_open())
	{
	for (int j=0;j<n;j++)
	{
		for (int i=0; i<3;i++)
			{
		outfile << data[i][j];
					if (i<2) outfile << " ";
```

```
                                        }
                        outfile << endl;
                }
                }
                else{ cout << "Error Output file is already created" << endl;}
                outfile.close();

                cout << "A-OK" << endl;
                //cin.get();
//          cin >> buffer; // Slow interface for PC compiler
        return 0;
}
```