SANDIA REPORT

SAND99-2169

Unlimited Release

Printed January 2000

Generating Complex Molecular Graphics Using Automated Programs that Work with Raster 3D

Derek T. Mehlhorn

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

RECEIVED

JAN 28 2000

OSTI

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (703) 605-6000
Web site: <http://www.ntis.gov/ordering.htm>

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND99-2169
Unlimited Release
Printed January 2000

Generating Complex Molecular Graphics Using Automated Programs that Work with Raster 3D

Derek T. Mehlhorn
Computational Biology and Materials Technology Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1111

Abstract

Two programs have been written in C++ to greatly automate the process of computer simulation visualization in most cases. These programs, *rasterize.C* and *tracker.C*, can be used to generate numerous images in order to create a video or still frames. In order to limit the amount of time and work involved in visualizing simulations, both of these programs have their own specific output formats. The first output format, from *rasterize.C*, is best suited for those who need only to visualize the actions of a single element, or elements that work on roughly the same time scale. The second format, from *tracker.C*, is best suited for simulations which involve multiple elements that work on different time scales and thus must be represented in a manner other than straight forward visualization.

Contents

Generating Complex Molecular Graphics Using Automated Programs That Work With Raster3D

Introduction.....	1
Program Design	1
Input Formats	1
Type and Color Definition	3
Size Definition	3
Determining the Number of Images to Generate.....	4
Graphical Design	5
Probable Sources of Errors	6
Figure 1: A Snapshot of a movie generated by <i>tracker.C</i>	7
Table 1: Summary of <i>rastize.C</i> and <i>tracker.C</i> Differences	8
References	9
Appendix A: List of Support Files for <i>rasterize.C</i> and <i>tracker.C</i>	10
Appendix B: Sample Input Files for <i>rasterize.C</i> and <i>tracker.C</i>	11
Appendix C: Function Explanations for both <i>rasterize.C</i> and <i>tracker.C</i>	12
Appendix D: Source Code for <i>rasterize.C</i>	14
Appendix E: Source Code for <i>tracker.C</i>	29

Generating Complex Molecular Graphics Using Automated Programs That Work With Raster 3D

Introduction

One can never truly comprehend the extent of one's computer simulation without being able to visualize the results. However, due to time constraints and the amount of effort usually involved in creating such visualizations, more often than not these simulations are never graphically represented. For exactly this reason two programs have been written that will greatly simplify and accelerate this visualization process for molecular simulations

A major goal of the two programs, *tracker.C* and *rasterize.C*, was to use C++ to enable a user to generate consistent graphics from large dump files in order to generate movies. A computer program that works to such an end is required due to the sheer size of a dump file containing information from almost any number of time-steps. A computer program is also desired because it saves the user time by finding and extracting important information from a file without an undue amount of formatting.

Program Design

One of the goals of these programs, was to allow a user to input large amounts of data for a large number of time-steps, with a minimal amount of formatting, and produce a specified number of images that can be converted into a movie. Although their primary purpose is to generate numerous images for use in movies, it is also very capable of rendering individual frames. To either end, video or still frame, each of these programs present different options in graphical representation.

Both of these programs accomplish their graphical objectives using similar input methods, and using the same graphics program, *Raster3d*[1]. *Raster3d* is a simple but powerful shareware program which enables the user to specify shapes, coordinates, size, and color through a series of input files. *Rasterize.C* and *tracker.C* manipulate these input files using data extracted from a given data file, thus rendering the desired graphics. Although *Raster3d* has many great features, there are also some large drawbacks that limit how general these programs can be. The major *Raster3d* related drawback within the *tracker.C* program is the inability for *Raster3d* to display text of any kind. To overcome this severe handicap one must prepare a separate image, where the comments are correctly placed, and then overlay them. This process is tedious and cannot be automated, so it is for this reason that the program *tracker.C* will be the more difficult of the two programs to be adapted for new simulations.

Input Formats

The first program, *rasterize.C*, takes a dump file which specifies atom position, time step, and simulation size. Input files for *rasterize.C* should be formatted as follows:

```
ITEM: BOX BOUNDS
      0.0000000000000000E+000    107.72173000000000
      0.0000000000000000E+000    107.72173000000000
      0.0000000000000000E+000    107.72173000000000
ITEM: TIMESTEP
      0
ITEM: NUMBER OF ATOMS
```

```

1
ITEM: ATOMS
980104 2 0.03669 0.01582 0.00120
ITEM: TIMESTEP
20
ITEM: NUMBER OF ATOMS
1
ITEM: ATOMS
980104 2 0.03697 0.01769 0.00199

```

Input files for *tracker.C* look like this:

```

ITEM: BOX BOUNDS
-42.00009999999747      42.00009999999747
-12.000000000000000    12.000000000000000
-12.000000000000000    12.000000000000000
ITEM: ID STRING
wltest_57.05
ITEM: NUMBER OF ATOMS
8537
ITEM: TIMESTEP
160000
ITEM: ATOMS
6669 1 0.10312 0.28906 0.35928
3434 1 0.04195 0.27004 0.22129
6052 1 0.14231 0.12134 0.33583
5434 1 0.12298 0.39656 0.49550

```

The most important difference between the input (dump) file for *rasterize.C* and *tracker.C* is that the input for *rasterize.C* includes a definition of the number of atoms at each time-step, whereas the input file for *tracker.C* does not. This difference will be addressed a little later. Another difference between these two inputs is that *tracker.C* includes a reference file name. This difference will also be examined further later on. Some other differences between these formats will have no impact on the programs at all. For example, extra information, such as re-definition of the box bounds, shouldn't have any impact on either program since it only looks for it at the beginning of the file. In order to avoid any unexpected problems it is recommended that the box definitions be put at the very beginning of the input file.

Both of these programs read through their input files linearly so the order of information must be made the same as above (with the few exceptions mentioned earlier) and remain so throughout the entire file. An important difference to note between *rasterize.C* and *tracker.C*, is that *rasterize.C* needs an input file that states the number of atoms at each time-step. *Tracker.C* on the other hand will read in atoms until it reads in a string containing "ITEM" (as is found before each new time step). It is recommended that the extra information required by *rasterize.C* be added when generating the input (dump) file if one plans to use either of these two programs. The reason for this is that although *tracker.C* doesn't need this information, it will not be hindered by it. Therefore the automatic addition of this small amount of information will make an input file readable by both programs.

The information for atoms is entered as follows:

```
ITEM: ATOMS
id      type    x-pos    y-pos    z-pos
```

where *id* is an atom's unique identification number, *type* is a number identifying an atom's type, *x-pos* is its position in relation to the x-axis, *y-pos* is its position in relation to the y-axis, and *z-pos* is its position in relation to the z-axis. These five pieces of information are the basis of each of the programs; without this information, the rendering of images with either program is not possible, therefore one must be certain that this information is formatted correctly.

When comparing the *support.data* files for both programs, one can see that two input (dump) files are provided for *tracker.C* and only one for *rasterize.C*. The reason for the second input (dump) file is specific to the simulation (the LADERA[2] grand canonical molecular dynamics code) that *tracker.C* was initially meant to visualize and can therefore be eliminated by replacing the file name with the string "none" (without quotes, all lowercase). If one is running a constant pressure simulation, such as LADERA, that requires pistons, the dump file which contains this data can be placed in the second position of the *tracker.C* input file. A problem with inserting a new dump file name is that it probably doesn't have the same format as the one the program was initially designed to read, so it is recommended that the second input file for *tracker.C* not be used unless the user knows what the input format should be. A sample piston dump file can be found in the file *example.log*.

Type and Color Definition

The "type" of an atom is what determines its color when represented graphically. *Tracker.C*'s color determination is built into the code and supports a maximum of 4 different types of atoms (types 2-5). Because of this, changing the color of the output requires modifying the code slightly. The color scheme is stored in an array called *rgb*, and it stores the color scheme in red, green, and blue (as does *Raster3d*). When a user wishes to modify the color of their atoms they need only modify the array's initialization. The color array is five rows by four columns, each row being a new type. It is important to note that the array determines the row to read from by taking the atom type and subtracting one. So if the atom type is 5 it will read from row 4.

As mentioned before, the color definitions for *tracker.C* are different from those of *rasterize.C*. *Tracker.C* receives its color scheme from the user input file *support.data*. This allows the user to more easily manipulate the colors of the images, however, *tracker.C* only supports two atom types. Therefore, if more atom types are required parts of *tracker.C* will need to be modified.

Size Definition

As for the definition of atom size, both programs receive the size of each atom (designated by type) from their user input files. The size input for each atom is recognized as its radius, so one should be careful not to input the diameter by accident. The size variable is treated similarly to color in both programs respectively, this *rasterize.C* will accept 4 and *tracker.C* will only accept 2.

Determining the Number of Images to Generate

Most of the information related to the image itself has already been mentioned above, however, since the primary purpose of these programs was to generate a large number of images in one run, information related to time and length must also be input. All the time information that is required by a given program should be put in the input file *support.data*. An initial *support.data* file should be provided with each of the programs, and can be found in **Appendix C**.

The program *rasterize.C* is the simplest of the two programs within which to designate time. *Rasterize.C* simply needs the total time, the amount of time dedicated to equilibration (if any), and the number of time-steps per frame. The first parameter (total number of time-steps) is self explanatory: it is the total number of time-steps that the program should run (including the equilibration period). The second parameter (neq) is for cases where a simulation has an equilibration period in which the atoms in the simulation are allowed to mix (this parameter should not be used if the equilibration period is not included in the dump file). If the user does not want to display this period of randomization then this parameter should be used, otherwise, if the user wants to visualize the entire simulation (everything in the dump file) this parameter should be set to the first time step in the file. The third parameter (step size) designates how many time-steps the program should skip before it generates a new image. Consider the following example: say a dump file contains information for every 100 time-steps, but the user only wants to visualize every 500 time steps. In order to make this adjustment the user need only change the step size (third parameter mentioned above) to 500.

The program *tracker.C* has a similar input format, but has much stricter requirements for input values. As will be discussed later, *tracker.C* generates a total of six different images, one image per specific time interval. This time interval was determined very simply by dividing the total amount of run time by five. Thus an image is printed at intervals of 20% of the total time. This fact complicates matters because the time interval between frames must be a factor of the total time divided by five, otherwise *tracker.C* will skip the time step at which it is suppose to print. In order to find out what 20% of the total run time is, we subtract the first time-step value in the dump file from the total number of time steps, as designated by *support.dat*, and divide by five. Then, in order to convert this number from time-steps to the number of image frames, the result of the previous step is then divided by the time-step interval between frames. To clarify consider an excerpt from *tracker.C*'s *support.data* file:

```
...
520000      Total number of time steps
6000        Desired number of time steps per frame
...
```

The data seen above is appropriate for *tracker.C* because 520,000 (total number of time-steps) minus 160,000 (first time-step in dump file) divided by five is equal to 72,000. So a new box image will be generated every 72,000 time frames, and since 72,000 divided by 6,000 (desired time interval between movie frames) is 12, we know that the program will read the correct time intervals to generate our images. In order to calculate the correct input values for *tracker.C* one can use this equation:

$$(\text{Total Time} - \text{Start time}) / 5 = \text{Time interval between box generation}$$

where the interval between movies frames (as defined in *support.data*) is a factor of the time interval between box generations (as is solved for above).

Something to note is that *tracker.C* doesn't allow equilibration information to be added to the beginning of the dump file unless the user wants this information to be visualized. Instead, *tracker.C*'s dump file should merely start at the time step at which the user wishes visualization to begin.

A complete list of input files, and examples of each file can be found in Appendix A.

Graphical Design

As mentioned earlier *tracker.C* and *rasterize.C* have very different ways of representing information graphically. *Rasterize.C* is a simple read-and-display program, which possesses status bars to assist the viewer. This program will only render spheres with up to four types (color and size definitions) and a simulation box. An example of the *rasterize.C* output is shown below.

Tracker.C on the other hand was designed to represent a simulation where two factors needed to be represented on two different time scales. Therefore, *tracker.C* creates a graph of atom density versus x-position, and, by the end of the program, six images representing the simulation system (atoms and simulation box) at different times. As was discussed in the section **Determining the Number of Images to Generate**, *tracker.C* produces a new image of the system at intervals of 20% of the total time. The graph (at present) is an average of all the time steps between each movie frame. So if we use the input file seen in the previous section, a movie frame is created every 6,000 time steps. Therefore, the graph in every movie frame is an average of the previous 6,000 time steps.

The graph in the program *tracker.C* will undoubtedly cause problems for any simulations other than the ones that it was initially meant to model (LADERA), therefore, the graph can be disabled by commenting out the function call for *ploter*, about line 329 of *tracker.C* (the actual line is marked by comments). The graph can be made to display other information, but that would require changing a fair amount of code.

As mentioned above, *rasterize.C* supports two status bars, one that represents the number of atoms, and one that represents time. Since the total time is user defined, that status bar is unlikely to require adjustment. However, the atoms bar may require being changing if the number of atoms is not supplied for every time-step contained in the dump file. For more information on this see **Input Formats**. One should remember that these status bars can be toggled on or off using the *support.data* file.

Tracker.C also has a few optional graphical features such as how the grand canonical molecular dynamics control volumes are displayed on individual system images. The three options for this feature are, none (0), transparent planes (1), rectangular outlines (2). In the sample output contained in **Figure 1**, transparent planes are used to represent the control boxes on each small image.

As mentioned briefly above, the text that appears on this image is not built into the code because text is not supported by *Raster3d*. This text was produced by overlaying the graphical image with a preset image containing text. Overlaying two images is done by the “combine -over” command, and is automatically written into the program scripts. In order to disable text completely, one must either change the file *labels.tiff* to a white image, or comment out the correct part of the function *scripter*. The correct code to comment out is commented within the previously mentioned function (see page 48 of **Appendix E**). However, the first method of eliminating comments is recommended because the combining script also performs other conversions which are required due to oddities in *Raster3d*. If one wishes to change the comments that are to be displayed on the images, one need only modify the image *labels.tiff* with any graphics program that supports text.

Probable Sources of Errors

Due to irregularities in dump files from one time-step to the next, certain parsing methods were used in *rasterize.C* that will very likely cause errors when working with new input files. The most probable error will be the occurrence of infinite *while* loops. This error may be caused by the fact that the “getline” command in C++ gets spaces before and after any text that appears on a given line. Currently, *rasterize.C* looks for a single space before key strings (those starting with “ITEM:”), therefore, if there are more or less spaces before these strings an infinite *while* loops will be entered. This problem can be fixed by changing some of the string comparison statements (add or subtracting spaces). The simplest way to avoid this problem is to only add one space at the beginning of a line, and have none afterwards.

Other errors may occur due to bad formatting within the dump file itself or the *support.data* files. See the formatting information above.

Figure 1

A snapshot of a movie generated from the LADERA grand canonical molecular dynamics code by *tracker.C*

Table 1: Summary of *rasterize.C* and *tracker.C* Differences

Subject	<i>rasterize.C</i>	<i>tracker.C</i>
Input Formats	See Pages 1-3	See Pages 1-3
Type and Color Definition	Program supports a maximum of 4 atom types, and color scheme is defined within code	Program supports a maximum of 2 atom types, and color is defined in <i>support.data</i>
Size Definition	Atom size is defined in <i>support.data</i>	Atom Size is defined in <i>support.data</i>
Determining the Number of Images	<p>Requires definition of total time, number of steps allotted for equilibration (neq), and number of time steps to skip between images.</p> <p>Accepts any step value which is a factor of the total - the neq</p>	<p>Requires definition of total time, and number of time steps to skip between images.</p> <p>Step value must be a factor of the equation $(\text{Total time} - \text{Start time}) / 5$</p>

References

1. Ethan, M.A. <http://www.bmsc.washington.edu/raster3d.html>
2. Heffelfinger, G.S., Thompson, A.P., Ford, D.M., Martin, M.G., *Gradient-Driven Diffusion of Multi-Atom Molecules Through Macromolecules and Membranes: LDRD 96-0021 Close-out Report*, SAND98-2696, 1999.

Appendix A

List of Support Files for *rasterize.C* and *tracker.C*

support.hda¹

This file contains information related to how *raster3d* generates its images (nothing should need to be modified in this file)

support.hdb¹

This file contains information about objects in *raster3d* and should also not need to be modified

support.mat¹

This file contains information on the orientation of each image. This file should not be modified by the user because it is a composite of the rotate files.

support.data²

This is the input file specific to both *rasterize.C* and *tracker.C*. See examples of both versions of the file in Appendix B.

rotate, rotate.dat, rotate.f¹

These three files contain information about the rotation of each image. To change the rotation of an image or how close or far away the view is, one can modify *rotate.dat*.

```
0.0          theta on [-180,18]
0.0          phi  on [-180,180]
0.0          psi  on [-180,180]
0.0 10.0 0.0  xt(3)
210.0        h
```

The first three values of *rotate.dat* (as shown above) control the roll of the image. The fourth line defines the center of the image, and the last value defines the zoom (smaller is closer, larger is farther away)

labels.tiff

This file as discussed at the end of **Graphical Design** is the image that contains any labels pertinent to images produced by *tracker.C*.

¹ These files are the same for both *rasterize.C* and *tracker.C*

² These files have the same general purpose for both programs but vary in design

Appendix B

Sample Input Files for *rasterize.C* and *tracker.C*

Example of *support.data* for *rasterize.C*

lammps_dump_0.01_1	File name
107.72173000000000	edgex
107.72173000000000	edgey
107.72173000000000	edgez
2000	neq
20	# of timesteps per frame
2020	total timesteps desired w/ neq
1.5	radius 1
1.5	radius 2
1.5	radius 3
1.5	radius 4
0.90	Status bar max length
1	0=hide status bar 1=show status bar

Example of *support.data* for *tracker.C*

lammps_dump_57.05	Main tracker.C source file
wltest_57.05.log	Source file for piston positions
520000	Total number of timesteps
6000	Desired number of timesteps per frame
.5	radius of atom type 1
1	radius of atom type 2
1.0 0.0 0.0	color of atom type 1
0.0 1.0 0.0	color of atom type 2
-18 -6	X-coor for control box 1
6 18	X-coor for control box 2
-6 6	Z-plane containing atoms to display
1	Features:0=nothing 1=planes 2=outlines

Appendix C

Function explanations for both *rasterize.C* and *tracker.C*

rasterize.C Functions

void make

A function that creates individual nodes for a linked list.

void remove

A function that removes individual nodes from a linked list.

void boxFrame

This function creates the simulation box visualization.

void scripter

This function writes and executes the scripts that renders images.

void Mover

This function cleans up after *rasterize.C* by moving and deleting specific files.

void Bars

This function creates the status bars.

void MakeBars

This function creates cylinders.

void MakeSphere

This function creates spheres.

int atomFind

This function searches to the last time step desired and returns the total number of atoms for use by the *Bars* function.

int myround

This function rounds numbers to the nearest integer.

tracker.C Functions

void make

A function that creates individual nodes for a linked list.

void remove

A function that removes individual nodes from a linked list.

void boxFrame

This function creates the simulation box visualization.

void ploter

This function creates the graph.

void pistons

This function visualizes pistons from information in a second dumpfile.

void MakeBars

This function creates cylinders.

void MakeSpheres

This function creates spheres.

void MakePlane

This function creates planes.

void scripter

This function writes and executes scripts that generate images as well as combines and converts the final products to visualize text.

void mover

This function cleans up after *tracker.C* by moving and removing specific files.

int myround

This function rounds numbers to the nearest integer.

Appendix D

Source Code for *rasterize.C*

```
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <iomanip.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct listNode
{
    int id, type;
    double x, y, z;
    listNode *next;

    listNode(int tempId, int tempType, double tempx, double tempy, double tempz,
    listNode *nxt);
};

listNode::listNode (int tempId, int tempType, double tempx, double tempy, double tempz,
listNode *nxt)
:id(tempId), type(tempType), x(tempx), y(tempy), z(tempz), next(nxt)
{}

typedef listNode* nodePtr;

struct listType
{
    nodePtr first;
    nodePtr last;
};

void make(int&, int&, double&, double&, double&, listType&);
void remove(listType &, int &);
void boxFrame(double&, double&, double&, double , double , double , char[20],
int &, int &);
void scripiter(int &, int);
void mover(int &, int);
void Bars(int&, int&, int&, int&, int&, double&);
void MakeBars(double&, double&, double&, double&, double&, double&, double&,
double&, double&, double&, char[50]);
void MakeSphere(double &, double &, double &, double &, double &, double &,
double &, char[50]);
int atomFind(int&, char[50]);
int myround(double&, double&);

//*****> END of function declaration <*****

void main()
{
```

```

char data[50] = "support.data";           //User input file name
char source[50];
char file[50];
char inputt[201];

int timestep, randomNum, total, look, desired;
int atomNum = 1, num = 1, Atoms = 0;
int kind, id, itype, pop;
int flag = 0;

double x,y,z, barsize;
double avx=0.0, avy=0.0, avz=0.0;
double edgex, edgey, edgez;
double radius[5], rgb[5][4];
double sphere_red, sphere_green, sphere_blue, rsphere;
double masx, masy, masz;
double sumx, sumy, sumz;
double delx, dely, delz;

masx = avx;
masy = avy;
masz = avz;

listType list;
nodePtr temp;
list.first = list.last = NULL;

//*****>printing and reading conditions

cout.precision(4);
cin.precision(4);

ifstream infile;
infile.open(data, ios::in);
if (!infile) cerr << "BAD1" << endl;
cout.setf(ios::fixed);

//*****>

infile >> source;
infile.getline(inputt, 200);
infile >> edgex;
infile.getline(inputt, 200);
infile >> edgey;
infile.getline(inputt, 200);
infile >> edgez;
infile.getline(inputt, 200);

infile >> total;
infile.getline(inputt, 200);

infile >> randomNum;
infile.getline(inputt, 200);

```

```

infile >> desired;
infile.getline(inputt, 200);

infile >> radius[1];
infile.getline(inputt, 200);
infile >> radius[2];
infile.getline(inputt, 200);
infile >> radius[3];
infile.getline(inputt, 200);
infile >> radius[4];
infile.getline(inputt, 200);

infile >> barsize;
infile.getline(inputt, 200);

infile >> flag;
infile.getline(inputt, 200);

infile.close();

rgb[1][1] = 1.0;
rgb[1][2] = 0.0;
rgb[1][3] = 0.0;

rgb[2][1] = 1.0;
rgb[2][2] = 0.0;
rgb[2][3] = 0.0;

rgb[3][1] = 0.0;
rgb[3][2] = 0.0;
rgb[3][3] = 1.0;

rgb[4][1] = 1.0;
rgb[4][2] = 1.0;
rgb[4][3] = 0.0;

//*****>

infile.open(source, ios::in);
if (!source) cerr << "BAD2" << endl;
infile.setf(ios::fixed);

while (strcmp(inputt, " ITEM: ATOMS"))
    infile.getline(inputt, 200);

infile >> inputt >> inputt;
infile >> avx >> avy >> avz;

infile.close();

//*****>

Atoms = atomFind(total, source);

```

```

look = randomNum;
timestep = 0;

while (timestep < total)
{
    sprintf(file, "rast.r3d");

    infile.open(source, ios::in);
    if (!source) cerr << "BAD3" << endl;
    infile.setf(ios::fixed);

    while (strcmp(inputt, " ITEM: ATOMS"))
infile.getline(inputt, 200);

    cout << "num -----> " << num << endl;

    while (!infile.eof() && timestep < look)
    {
while (strcmp(inputt, " ITEM: TIMESTEP"))
infile.getline(inputt, 200);

        infile >> timestep;
        infile.getline(inputt, 200);
    }

    if (timestep > look)
        cerr << "WARNING: timestep has surpassed given" << endl;

    while (strcmp(inputt, " ITEM: NUMBER OF ATOMS"))
        infile.getline(inputt, 200);

    infile >> atomNum;

    while (strcmp(inputt, " ITEM: ATOMS"))
        infile.getline(inputt, 200);

    cout << "Reading..." << endl;
    for (pop=1; pop < atomNum; pop++)
    {
infile >> id;
infile >> kind;
infile >> x >> y >> z;

        if (x >= 0.5) x -= 1;
        if (y >= 0.5) y -= 1;
        if (z >= 0.5) z -= 1;

        x*=edgex;
        y*=edgey;
        z*=edgez;

        sumx += x;
        sumy += y;
        sumz += z;
    }
}

```

```

delx = x - masx;
dely = y - masy;
delz = z - masz;

x = x - edgex * myround(delx, edgex);
y = y - edgey * myround(dely, edgey);
z = z - edgez * myround(delz, edgez);

    make(id,kind,x,y,z,list);
}
infile.close();

sumx /= pop;
sumy /= pop;
sumz /= pop;
temp = list.first;

while (temp != NULL)
{
itype = temp->type;
rsphere = radius[itype-1];
sphere_red = rgb[itype-1][1];
sphere_green = rgb[itype-1][2];
sphere_blue = rgb[itype-1][3];

MakeSphere(temp->x, temp->y, temp->z, rsphere, sphere_red, sphere_green,
sphere_blue, file);
list.first = list.first->next;
remove(list, temp->id);
temp = list.first;
}

boxFrame(edgex, edgey, edgez, avx, avy, avz, file, atomNum, timestep);
sprintf(file, "status.r3d");

if (flag == 1)
    Bars(timestep, atomNum, total, randomNum, Atoms, barsize);
scripter(num, flag);
num++;

masx = sumx;
masy = sumy;
masz = sumz;

    look += desired;
}
mover(num, flag);
}

//*****>MYROUND FUNC-
TION<*****

int myround(double &del, double &edge)

```

```

{
    double val = 0.0;
    int num = 0;
    double dec = 0.0;

    val = del/edge;
    num = int(val);
    dec = val - num;

    if (dec == 0)
        return num;
    else if (dec < 0){
        if (dec >= -.5)
            return num;
        else if (dec < -.5)
            return num-1;
    }
    else if (dec > 0){
        if (dec >= .5)
            return num+1;
        else if (dec < .5)
            return num;
    }
    cerr << "ERROR IN ROUNDING" << endl;
    return 0;
}

//*****>MAKE<*****

void make(int &id, int &type, double &x, double &y, double &z, listType &list)
{
    if (list.first == NULL)
        list.first = list.last = new listNode(id,type,x,y,z,NULL);
    else{
        list.last->next = new listNode(id,type,x,y,z,NULL);
        list.last = list.last->next;
    }
}

//*****>BOX-
FRAME<*****

void boxFrame(double &edgex, double &edgex, double &edgez, double avx, double
avy, double avz, char write[20], int &atomNum, int&timestep)
{
    double orx, ory, orz;

    orx = avx - (.5 * edgex);
    ory = avy - (.5 * edgex);
    orz = avz - (.5 * edgez);

    double cube_red, cube_green, cube_blue;
    double ax_red, ax_green, ax_blue;
    double rcyl;

```



```

double rxj, ryj, rzj;

double ix, iy, iz, kx, ky, kz;

cube_red=1.0;
cube_green=1.0;
cube_blue=1.0;

rcyl= .27;

//*****>

rxj = orx + 5;
ryj = ory;
rzj = orz;

ax_red=1.0;
ax_green=0.0;
ax_blue=0.0;
MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, ax_red, ax_green, ax_blue,
write);

//*****>

rxj = orx;
ryj = ory + 5;
rzj = orz;

ax_red=0.0;
ax_green=1.0;
ax_blue=0.0;
MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, ax_red, ax_green, ax_blue,
write);

//*****>

rxj = orx;
ryj = ory;
rzj = orz + 5;

ax_red=0.0;
ax_green=0.0;
ax_blue=1.0;
MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, ax_red, ax_green, ax_blue,
write);

//*****>

rcyl = .25;

rxj = orx + edgex;
ryj = ory;
rzj = orz;
MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, cube_red, cube_green,
cube_blue, write);

```

```

//*****>

    rxj = orx;
    ryj = ory + edgey;
    rzj = orz;
    MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, cube_red, cube_green,
cube_blue, write);

//*****>

    rxj = orx;
    ryj = ory;
    rzj = orz + edgez;
    MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, cube_red, cube_green,
cube_blue, write);

//*****>

    ix = orx + edgex;
    iy = ory;
    iz = orz;

    kx = orx + edgex;
    ky = ory;
    kz = orz + edgez;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);

//*****>

    kx = orx + edgex;
    ky = ory + edgey;
    kz = orz;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);

//*****>

    ix = orx + edgex;
    iy = ory;
    iz = orz + edgez;

    kx = orx;
    ky = ory;
    kz = orz + edgez;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);

//*****>

    kx = orx + edgex;
    ky = ory + edgey;
    kz = orz + edgez;

```

```

    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);

//*****>

    ix = orx;
    iy = ory + edgey;
    iz = orz + edgez;

    kx = orx;
    ky = ory + edgey;
    kz = orz;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);

//*****>

    kx = orx;
    ky = ory;
    kz = orz + edgez;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);

//*****>

    kx = orx + edgex;
    ky = ory + edgey;
    kz = orz + edgez;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);

//*****>

    ix = orx + edgex;
    iy = ory + edgey;
    iz = orz;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);

//*****>

    kx = orx;
    ky = ory + edgey;
    kz = orz;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
}

//*****>BARS FUNC-
TION<*****

void Bars(int &timestep, int &atomNum, int &total, int &randomNum, int &Atoms,
double &barsize)
{

```

```

char file[50];
sprintf(file, "status.r3d");

ofstream out;
out.open(file);
out << " ";
out.close();

double atomScale, timeScale;

double ymove = 0.0;
double zmove = 0.0;
double xmove = 0.0;

double xstart = -0.45;
double ystart = -.45;
double zstart = 0.0;

double ix, iy, iz, kx, ky, kz, rcyl;
double cube_red, cube_green, cube_blue;

atomScale = double(Atoms) / double(barsize);
timeScale = (double(total) - double(randomNum)) / double(barsize);

//*****>ATOM CONTROL

rcyl = 0.0075;
xmove = barsize;

ix = xstart;
iy = ystart;
iz = zstart;

kx = xstart + xmove;
ky = ystart + ymove;
kz = zstart + zmove;

cube_red = 1.0;
cube_green = 1.0;
cube_blue = 1.0;

MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
file);

//*****>ATOM STATUS

rcyl = .008;
xmove = atomNum / atomScale;

kx = ix + xmove;
ky = iy + ymove;
kz = iz + zmove;

cube_red = 1.0;

```

```

cube_green = 0.0;
cube_blue = 0.0;

MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
file);

//*****> TIME CONTROL

rcyl = 0.0075;
ystart += -.02;
xmove = barsize;

ix = xstart;
iy = ystart;
iz = zstart;

kx = xstart + xmove;
ky = ystart + ymove;
kz = zstart + zmove;

cube_red = 1.0;
cube_green = 1.0;
cube_blue = 1.0;

MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
file);

//*****>TIME STATUS

rcyl = .008;
xmove = (double (timestep) - double(randomNum)) / timeScale;

kx = xstart + xmove;
ky = ystart + ymove;
kz = zstart + zmove;

cube_red = 0.0;
cube_green = 0.0;
cube_blue = 1.0;

MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
file);

//*****>EYE CANDY

ix = xstart - .015;
iy = ystart + .02;
iz = zstart;

cube_red = 1.0;
cube_green = 0.0;
cube_blue = 0.0;

rcyl = .008;

```

```

MakeSphere(ix, iy, iz, rcyl, cube_red, cube_green, cube_blue, file);

//*****>

ix = xstart - .025;
iy = ystart + .005;
iz = zstart;

kx = xstart - .005;
ky = iy;
kz = iz;

cube_red = 0.0;
cube_green = 0.0;
cube_blue = 1.0;

rcyl = .003;

MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
file);

//*****>

ix = xstart - .0145;
iy = ystart + .005;
iz = zstart;

kx = ix;
ky = ystart - .007;
kz = iz;

rcyl = .003;

MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
file);
}

//*****>MAKEBARS FUNC-
TION<*****

void MakeBars(double &ix, double &iy, double &iz, double &kx, double &ky, dou-
ble &kz, double &rcyl, double &red, double &green, double &blue, char file[50])
{
    ofstream outfile;
    outfile.open(file, ios::app);

    outfile <<5<< endl;
    outfile.setf(ios::scientific);
    outfile
    <<ix<<setw(15)<<iy<<setw(15)<<iz<<setw(15)<<rcyl<<setw(15)<<kx<<setw(15)<<ky<<
    setw(15)<<kz<<setw(15)<<rcyl;
    outfile.unsetf(ios::scientific);
    outfile.setf(ios::fixed);

```

```

    outfile <<setw(12)<<red<<setw(10)<<green<<setw(10)<<blue<< endl;
    outfile.unsetf(ios::fixed);

    outfile.close();
}

```

```

//*****>ATOMFIND FUNC-
TION<*****

```

```

int atomFind(int &total, char source[50])
{
    int time = 0;
    int number = 0;
    char inputt[201];
    ifstream in;

    cout << "total time == " << total << endl;

    in.open(source, ios::in);
    if(!source) cerr << "ATOM READING ERROR" << endl;

    while (time < total)
    {
        in.getline(inputt, 200);

        while (strcmp(inputt, " ITEM: TIMESTEP"))
            in.getline(inputt, 200);
        in >> time;
    }

    while (strcmp(inputt, " ITEM: NUMBER OF ATOMS"))
        in.getline(inputt, 200);

    in >> number;
    cout << "number of atoms == " << number << endl;
    return number;
}

```

```

//*****>SCRIPTER FUNC-
TION<*****

```

```

void scripiter(int &select, int flag)
{
    char file[50], name[50], image[50], normal[50];
    ofstream out;

    sprintf(file, "rscript");
    sprintf(name, "rast.r3d");
    sprintf(normal, "status.r3d");
    sprintf(image, "image-%03d", select);
    out.open(file);

    out << "#!/bin/csh -f" << endl;
    out << "cat rotate.dat | ./rotate >! support.mat" << endl;
}

```

```

    if (flag == 0){
        out << "cat support.hda support.mat support.hdb ";
        out << name;
        out << " | render -tiff " << image << endl;
    }
    else if (flag == 1){
        out << "cat support.hda support.mat support.hdb " << name << " >!
stuff.r3d" << endl;
        out << "normal3d < stuff.r3d > norm.r3d" << endl;
        out << "cat norm.r3d " << normal << " | render -tiff " << image << endl;
    }

    out.close();

    sprintf(file, "chmod u=wrx rscript");
    system(file);
    system("rscript");

    sprintf(file, "cor.script");
    out.open(file);
    out << "#!/bin/csh -f" << endl;
    out << "convert -flip " << image << " " << image << endl;
    out.close();
    system("chmod u=wrx cor.script");
    system("cor.script");
}

//*****>MAKESPHERE FUNC-
TION<*****

void MakeSphere(double &x, double &y, double &z, double &rsphere, double &red,
double &green, double &blue, char file[50])
{
    ofstream outfile;
    outfile.open(file, ios::app);

    outfile << 2 << endl;
    outfile.precision(4);
    outfile.setf(ios::scientific);
    outfile << setw(5) << x << setw(12) << y << setw(12) << z << setw(12) <<
rsphere;
    outfile.unsetf(ios::scientific);
    outfile.setf(ios::fixed);
    outfile << setw(10) << red << setw(10) << green << setw(10) << blue << endl;
    outfile.unsetf(ios::fixed);
}

//*****>MOVER FUNC-
TION<*****

void mover(int &num, int flag)
{
    char command[200];

```



```

cout << "Cleaning Up..." << endl;
for (int p = 1; p < num; p++)
{
    sprintf(command, "mv image-%03d pics/", p);
    system(command);
}
if (flag == 1)
    system("rm rscript rast.r3d status.r3d norm.r3d stuff.r3d cor.script");
else if (flag == 0)
    system("rm rscript rast.r3d cor.script");
}

//*****>LINKED LIST REMOVE FUNCTION<*****

void remove(listType &list, int &id)
{
    nodePtr temp, marker, back;
    if (list.first == NULL)
        cout << endl;
    else{
        if (list.first->id == id){
            marker = list.first;

            if (list.first == list.last)
                list.first = list.last = NULL;
            else
                list.first = list.first->next;

            delete marker;
        }
        else {
            temp = list.first;

            while (temp != NULL && temp->id != id)
            {
                back = temp;
                temp = temp->next;
            }
            if (temp != NULL){
                marker = temp;
                back->next = temp->next;

                if (temp == list.last)
                    list.last = back;

                delete marker;
            }
        }
    }
}

```

Appendix E

Source Code for *tracker.C*

```
#include <iostream.h>
#include <string>
#include <fstream.h>
#include <math.h>
#include <iomanip.h>
#include <stdio.h>
#include <stdlib.h>

struct listNode
{
    int id, type;
    double x, y, z;
    listNode *next;

    listNode(int tempId, int tempType, double tempx, double tempy, double tempz,
    listNode *nxt);
};

listNode::listNode (int tempId, int tempType, double tempx, double tempy, double tempz, listNode *nxt)
: id(tempId), type(tempType), x(tempx), y(tempy), z(tempz), next(nxt)
{}

typedef listNode* nodePtr;

struct listType
{
    nodePtr first;
    nodePtr last;
};

//*****<Linked List functions>
void make(int&, int&, double&, double&, double&, listType&);
void remove(listType &, int &);

//*****<Graphical Options>
void boxFrame(double&, double&, double&, double , double , double , char[20],
int &, int &, double, double, double, double, int);
void ploter(double&, double&, double&, double&, double&, double&, listType,
double, double, double, double, double*, double*, int, double&, double&, double&, double&, double&);
void pistons(char[50], char[50], int, double, double, double, double, double, double, double);

//*****<Shape functions>
void MakeBars(double&, double&, double&, double&, double&, double&, double&, double&, double&, double&, char[50]);
```

```

void MakeSpheres(double &, double &, double&, double &, double &, double &,
double &, char[50]);
void MakePlane(double &, double &, double &, double &, double &, double &, dou-
ble &, double &, double &, double &, double &, double&, double &,
double &, double &, char[50]);

//*****<>
void scripter(int &);
void mover(int &);
int myround(double &);

void main()
{
    char file[50] = "support.data";
    char source[50];
    char garbage[201];
    char write[50] = "trac.r3d";
    char place[50];

    double endx, endy, endz;
    double orx, ory, orz;
    double edgex, edgy, edgez;
    double x, y, z;
    double radius[3], rgb[3][4];
    double red, green, blue;
    double size, plane1, plane2;
    double con11, con12, con21, con22;
    double tox, toy, toz;
    double rr, rg, rb, gr, gg, gb;
    double vol = 0.0;

    int atomNum, timestep, total, step, num;
    int id, type, look, find=0, start, period, store, turn;
    int flagged, stuff, def;
    long int position = 1;

    ofstream outfile;
    outfile.precision(4);

    listType list;
    nodePtr temp;
    list.first = list.last = NULL;
//*****>
//*****>Setting up transparent Surfaces
//*****>
    outfile.open("con.r3d");
    outfile << 8 << endl;
    outfile << 17.0 << setw(10) << 0.6 << setw(10);
    outfile << 0.0 << setw(10) << 0.0 << setw(10) << 1.0 << setw(10);
    outfile << 0.5 << setw(10) << 0 << setw(10) << 0 << setw(10) << 0 <<
setw(10)<< 0 << endl;
    outfile.close();
//*****>
//*****>Reading from Input File

```

```

//*****>
ifstream infile;
infile.open(file, ios::in);
if (!infile) cerr << "ERROR TYPE 1-1" << endl;

infile >> source;
infile.getline(garbage, 200);

infile >> place;
infile.getline(garbage, 200);

infile >> total;
infile.getline(garbage, 200);

infile >> step;
infile.getline(garbage, 200);

infile >> radius[1];
infile.getline(garbage, 200);
infile >> radius[2];
infile.getline(garbage, 200);

infile >> rgb[1][1] >> rgb[1][2] >> rgb[1][3];
infile.getline(garbage, 200);
infile >> rgb[2][1] >> rgb[2][2] >> rgb [2][3];
infile.getline(garbage, 200);

rr = rgb[1][1]; rg = rgb[1][2]; rb = rgb[1][3];
gr = rgb[2][1]; gg = rgb[2][2]; gb = rgb[2][3];

infile >> con11 >> con12;
infile.getline(garbage, 200);
infile >> con21 >> con22;
infile.getline(garbage, 200);

infile >> plane1 >> plane2;
infile.getline(garbage, 200);

infile >> flagged;
infile.getline(garbage, 200);

infile.close();
//*****>
//*****>Getting starting information
//*****>
infile.open(source, ios::in);
if (!source) cerr << "ERROR TYPE 1-2" << endl;
infile.setf(ios::fixed);
//*****>
position = infile.tellg();
while (strcmp(garbage, " ITEM: TIMESTEP"))
    infile.getline(garbage, 200);
infile >> look; //Getting initial timestep to read until

```

```

infile.getline(garbage, 200);
while(strcmp(garbage, " ITEM: TIMESTEP"))
    infile.getline(garbage, 200);
infile >> def;

def -= look;
start = look;
look += step;
//*****>
infile.seekg(position);
while (strcmp(garbage, " ITEM: BOX BOUNDS"))
    infile.getline(garbage, 200);

infile >> orx >> endx;
infile >> ory >> endy;    //Reading in Box Bounds
infile >> orz >> endz;
//*****>
infile.close();
//*****>
ory += 50;
endy += 50;    //Correcting Y Position
//*****>
edgex = endx - orx;
edgex = endy - ory;    //Calculating Bound Lengths
edgex = endz - orz;
//*****>
tox = orx;
toy = ory;    //Initializing Turn Box Origin Variables
toz = orz;
//*****>
con11 += (.5*edgex);
con12 += (.5*edgex);    //Correcting Control Box Coordinates
con21 += (.5*edgex);
con22 += (.5*edgex);
//*****>
//*****>Defining arrays for graph coordinates
//*****>
stuff = myround(edgex);

double rnumbers[stuff], gnumbers[stuff];
double tempr[stuff], tempg[stuff];
double *rp, *gp;

for (int wipe = 0; wipe < stuff; wipe++)
{
    rnumbers[wipe] = 0;
    gnumbers[wipe] = 0;
    tempr[wipe] = 0;
    tempg[wipe] = 0;
}

rp = rnumbers;
gp = gnumbers;
//*****>

```

```

    timestep = start;
//*****>
//*****>Main operations and writing to raster3d files
//*****>
    outfile.open(write);
    outfile << " ";
    outfile.close();

    num = 1;                                //Defines name of each image
    store = (total - start) / 5;           //Determines when to render a new box
    vol = (edgex * edgez * 1);             //Volume of column
    int compare = 0;
    int marker = 0;
    int val = 0;

    while (timestep < total)
    {
        cout << endl;
        cout << "Generating Image Number-----> " << num << endl;
        turn = (step / def);
        cout << "Reading..." << endl;

        cout << "time-> " << timestep + step << endl;

        for (int key = turn; key > 0; key--)
        {
            infile.open(source, ios::in);
            infile.seekg(position, ios::beg);

            while (strcmp(garbage, "TIMESTEP"))
                infile >> garbage;

            infile >> timestep;

            if (key == turn){
                if (timestep == start)
                    period = 1;
                else
                    period = store;
            }
            else
                period = -1;

            while(strcmp(garbage, "ATOMS"))
                infile >> garbage;

            compare = (timestep - start);

            if (compare == 0)
                compare = 0;
            else if (period == -1)
                compare = 1;
            else
                compare = compare % period;

```

```

if (compare == 0)
{
    marker = 1;
    while(strcmp(garbage, "ITEM:"))
    {
infile >> id;
infile >> type;
infile >> x;
infile >> y;
infile >> z;

x *= edgex;
y *= edgey;
z *= edgez;
make(id, type, x, y, z, list);

val = myround(x);

if (val>= 0 && val<= stuff)
{
    if (type == 1) tempr[val] += 1;
    else if (type == 2) tempg[val] += 1;
}

position = infile.tellg();
infile >> garbage;
infile.seekg(position, ios::beg);
    }
    position = infile.tellg();
}
else
{
    while (strcmp(garbage, "ITEM:"))
    {
infile >> id;
infile >> type;
infile >> x;
infile >> y;
infile >> z;

x *= edgex;

        val = myround(x);
if (val >= 0 && val <= stuff)
{
    if (type == 1) tempr[val] += 1;
    else if (type == 2) tempg[val] += 1;
}
position = infile.tellg();
infile >> garbage;
infile.seekg(position, ios::beg);
    }
    position = infile.tellg();
}

```

```

    }

    for (int dens = 0; dens < stuff; dens++)
    {
        tempr[dens] /= vol;
        tempg[dens] /= vol;
    }

    for (int finish = 0; finish < stuff; finish++)
    {
        rnumbers[finish] += tempr[finish];
        gnumbers[finish] += tempg[finish];
        tempr[finish] = 0;
        tempg[finish] = 0;
    }
    infile.close();
}

/**This funtion creates the Graph**
/*****>
    ploter(orz, ory, orz, edgex, edgey, edgez, list, con11, con12, con21,
con22, rp, gp, turn, rr, rg, rb, gr, gg, gb);
/*****>
/**                                     **

    for (int bleach = 0; bleach < stuff; bleach++){
        rnumbers[bleach] = 0;
        gnumbers[bleach] = 0;
    }

    if (marker == 1){
        find++;
        cout << "Boxes Generated -----< " << find << "
>" << endl;

        if (find == 1 || find == 2 || find == 3){
            tox = orx - (.5 * edgex) - 10;
            toy = ory - (40 * find);
            toz = orz;
        }
        else if (find == 4 || find == 5 || find == 6){
            tox = orx + (.5 * edgex);
            toy = ory - (40 * (find-3));
            toz = orz;
        }
        temp = list.first;

        while (temp != NULL)
    {
        type = temp->type;
        size = radius[type];
        red = rgb[type][1];
        green = rgb[type][2];
        blue = rgb[type][3];
    }
}

```



```

    x = temp->x + tox;
    y = temp->y + toy;
    z = temp->z + toz;

    if (z >= plane1 && z <= plane2)
        MakeSpheres(x, y, z, size, red, green, blue, write);

    list.first = list.first->next;
    remove(list, temp->id);
    temp = list.first;
}

    boxFrame(edgeX, edgeY, edgeZ, tox, toy, toz, write, atomNum, timestep,
con11, con12, con21, con22, flagged);
    turn = (timestep - start) / def;
    if (strcmp(place, "none"))
        pistons(place, write, turn, tox, toy, toz, edgeX, edgeY, edgeZ);
    }
    cout << "Rendering..." << endl;

    scripter(num);
    num++;
    look += step;
    marker = 0;
}
    cout << "Cleaning up..." << endl;
    mover(num);
}

//*****>MAKE<*****

void make(int &id, int &type, double &x, double &y, double &z, listType &list)
{
    if (list.first == NULL)
        list.first = list.last = new listNode(id,type,x,y,z,NULL);
    else{
        list.last->next = new listNode(id,type,x,y,z,NULL);
        list.last = list.last->next;
    }
}

//*****>BOX-
FRAME<*****

void boxFrame(double &edgeX, double &edgeY, double &edgeZ, double orx, double
ory, double orz, char write[20], int &atomNum, int&timestep, double con11,
double con12, double con21, double con22, int flagged)
{
    double cube_red, cube_green, cube_blue;
    double ax_red, ax_green, ax_blue;
    double rcyl;

```

```

double rxj, ryj, rzj;
double rxk, ryk;
double ix, iy, iz, kx, ky, kz;

cube_red=1.0;
cube_green=1.0;
cube_blue=1.0;

rcyl= .27;
//*****>
rxj = orx + 5;
ryj = ory;
rzj = orz;

ax_red=1.0;
ax_green=0.0;
ax_blue=0.0;
MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, ax_red, ax_green, ax_blue,
write);
//*****>
rxj = orx;
ryj = ory + 5;
rzj = orz;

ax_red=0.0;
ax_green=1.0;
ax_blue=0.0;
MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, ax_red, ax_green, ax_blue,
write);
//*****>
rxj = orx;
ryj = ory;
rzj = orz + 5;

ax_red=0.0;
ax_green=0.0;
ax_blue=1.0;
MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, ax_red, ax_green, ax_blue,
write);
//*****>
rcyl = .25;

rxj = orx + edgex;
ryj = ory;
rzj = orz;
MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, cube_red, cube_green,
cube_blue, write);
//*****>
rxj = orx;
ryj = ory + edgey;
rzj = orz;
MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, cube_red, cube_green,
cube_blue, write);
//*****>

```

```

    rxj = orx;
    ryj = ory;
    rzj = orz + edgez;
    MakeBars(orx, ory, orz, rxj, ryj, rzj, rcyl, cube_red, cube_green,
cube_blue, write);
//*****>
    ix = orx + edgex;
    iy = ory;
    iz = orz;

    kx = orx + edgex;
    ky = ory;
    kz = orz + edgez;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
//*****>
    kx = orx + edgex;
    ky = ory + edgey;
    kz = orz;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
//*****>
    ix = orx + edgex;
    iy = ory;
    iz = orz + edgez;

    kx = orx;
    ky = ory;
    kz = orz + edgez;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
//*****>
    kx = orx + edgex;
    ky = ory + edgey;
    kz = orz + edgez;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
//*****>
    ix = orx;
    iy = ory + edgey;
    iz = orz + edgez;

    kx = orx;
    ky = ory + edgey;
    kz = orz;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
//*****>
    kx = orx;
    ky = ory;
    kz = orz + edgez;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
//*****>

```

```

    kx = orx + edgex;
    ky = ory + edgey;
    kz = orz + edgez;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
//*****>
    ix = orx + edgex;
    iy = ory + edgey;
    iz = orz;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
//*****>
    kx = orx;
    ky = ory + edgey;
    kz = orz;
    MakeBars(ix, iy, iz, kx, ky, kz, rcyl, cube_red, cube_green, cube_blue,
write);
//*****>
//*****>Transparent Control Boxes
//*****>
    double trans = 0.9;      //<--Degree of Transparency
//*****>
    if (flagged != 0)
    {
        iz = orz + edgez+1;

        ix = orx + con11;
        iy = ory + edgey;

        kx = orx + con12;
        ky = ory + edgey;

        rxj = kx;
        ryj = ory;

        rxk = ix;
        ryk = ory;

        if (flagged == 1){
cube_red = 1.0;
cube_green = 1.0;
cube_blue = 1.0;
MakePlane(ix, iy, iz, kx, ky, iz, rxj, ryj, iz, rxk, ryk, iz, trans, cube_red,
cube_green, cube_blue, write);
        }
        else if (flagged == 2){
rcyl = .5;
cube_red = 0.0;
cube_green = 0.0;
cube_blue = 0.0;
MakeBars(ix, iy, iz, kx, ky, iz, rcyl, cube_red, cube_green, cube_blue, write);
MakeBars(kx, ky, iz, rxj, ryj, iz, rcyl, cube_red, cube_green, cube_blue,
write);

```

```

MakeBars(rxj, ryj, iz, rxk, ryk, iz, rcyl, cube_red, cube_green, cube_blue,
write);
MakeBars(rxk, ryk, iz, ix, iy, iz, rcyl, cube_red, cube_green, cube_blue,
write);
    }
    //*****>
        iz = orz + edgez+1;

        ix = orx + con21;
        iy = ory + edgey;

        kx = orx + con22;
        ky = ory + edgey;

        rxj = kx;
        ryj = ory;

        rxk = ix;
        ryk = ory;

        if (flagged == 1){
cube_red = 1.0;
cube_green = 1.0;
cube_blue = 1.0;
MakePlane(ix, iy, iz, kx, ky, iz, rxj, ryj, iz, rxk, ryk, iz, trans, cube_red,
cube_green, cube_blue, write);
        }
        else if (flagged == 2){
rcyl = .5;
cube_red = 0.0;
cube_green = 0.0;
cube_blue = 0.0;
MakeBars(ix, iy, iz, kx, ky, iz, rcyl, cube_red, cube_green, cube_blue, write);
MakeBars(kx, ky, iz, rxj, ryj, iz, rcyl, cube_red, cube_green, cube_blue,
write);
MakeBars(rxj, ryj, iz, rxk, ryk, iz, rcyl, cube_red, cube_green, cube_blue,
write);
MakeBars(rxk, ryk, iz, ix, iy, iz, rcyl, cube_red, cube_green, cube_blue,
write);
        }
    }
}

//*****>PIS-
TONS<*****
void pistons(char place[50], char location[50], int turn, double orx, double
ory, double orz, double edgex, double edgey, double edgez)
{
    ifstream in;
    in.open(place, ios::in);
    char crap[201];
    double xpos1, xpos2;
    int go = 0;
    double red, green, blue;

```

```

double newy, newz;

in >> crap;
while (go < turn)
{
    while (strcmp(crap, "-----"))
    {
in.getline(crap, 200);
in >> crap;
    }
    go++;
    in.getline(crap, 200);
}

for (int p = 0; p < 5; p++){
    in.getline(crap, 200);
}

while (strcmp(crap, "X_Field"))
    in >> crap;

in >> crap;
in >> xpos1;

while (strcmp(crap, "X_Field"))
    in >> crap;

in >> crap;
in >> xpos2;

in.close();

newy = ory + edgey;
newz = orz + edgez;

xpos1 += (.5 * edgex);
xpos2 += (.5 * edgex);

xpos1 += orx;
xpos2 += orx;

red = 0.0;
green = 0.0;
blue = 1.0;

double trans = 0.0;
MakePlane(xpos1, newy, orz, xpos1, newy, newz, xpos1, ory, newz, xpos1, ory,
orz, trans, red, green, blue, location);
MakePlane(xpos2, newy, orz, xpos2, newy, newz, xpos2, ory, newz, xpos2, ory,
orz, trans, red, green, blue, location);
}

//*****>MAKE-
PLANE<*****

```

```

void MakePlane(double &x, double &y, double &z, double &x2, double &y2, double
&z2, double &x3, double &y3, double &z3, double &x4, double &y4, double &z4,
double &trans, double &red, double &green, double &blue, char file[50])
{
    ofstream outfile;
    outfile.open(file, ios::app);

    if (trans > 0 && trans <= 1)
    {
        outfile << "#" << endl;
        outfile << 8 << endl;
        outfile << 17.0 << setw(10) << 0.6 << setw(10) << red << setw(10) << green
<< setw(10) << blue << setw(10)
        << trans << setw(10) << 0 << setw(10) << 0 << setw(10) << 0 << setw(10)
<< 0 << endl;
    }

    outfile << 1 << endl;
    outfile.setf(ios::scientific);
    outfile << x << setw(15) << y << setw(15) << z << setw(15);
    outfile << x2 << setw(15) << y2 << setw(15) << z2 << setw(15);
    outfile << x3 << setw(15) << y3 << setw(15) << z3 << setw(15);
    outfile.unsetf(ios::scientific);
    outfile.setf(ios::fixed);
    outfile << red << setw(10) << green << setw(10) << blue << endl;
    outfile.unsetf(ios::fixed);

    outfile << 1 << endl;
    outfile.setf(ios::scientific);
    outfile << x << setw(15) << y << setw(15) << z << setw(15);
    outfile << x4 << setw(15) << y4 << setw(15) << z4 << setw(15);
    outfile << x3 << setw(15) << y3 << setw(15) << z3 << setw(15);
    outfile.unsetf(ios::scientific);
    outfile.setf(ios::fixed);
    outfile << red << setw(10) << green << setw(10) << blue << endl;
    outfile.unsetf(ios::fixed);

    if (trans > 0 && trans <= 1)
        outfile << 9 << endl;

    outfile.close();
}

//
*****>PLOTTER<*****
*****
void ploter(double &orx, double &ory, double &orz, double &edgex, double
&edgex, double &edgez, listType list, double con11, double con12, double con21,
double con22, double *rp, double *gp, int turn, double &r1, double &g1, double
&b1, double &r2, double &g2, double &b2)
{
    cout << "graphing..." << endl;
    double enx, eny;

```

```

double dex, dey, stdz;
double newEdgex, newEdgey;
double size, correctionx, correctiony, corr, sum;
double newx, newy;
double oldx, oldy;
double trans = 0.0;
double red, green, blue;
//int val;
// nodePtr temp = list.first;

char name[50] = "graph.r3d"; //-->Output File Name
char other[50] = "con.r3d";
//*****>
corr = .4;
correctionx = .4 * edgex; //-->Increases x-axis by "correctionx" in each
direction
correctiony = .4 * edgey; //-->Increases y-axis by "correctiony" in positive
direction only
//*****>
size = .5;
//*****>
ofstream out;
out.open(name);
out << " ";
out.close();

out.open(other);
out << "#" << endl;
out.close();
//*****>
dex = orx - correctionx; //-->Left Most corner of the Graphing box (X)
dey = ory; //-->Left Most Corner of the Graphing box (Y)
stdz = orz; //-->Constant Z because of 2-D view

newEdgex = edgex + (2 * correctionx); //-->X-Edge Size for Graphing Box
newEdgey = edgey + (2 * correctiony); //-->Y-Edge Size for Graphing Box
//*****>
//*****>Box Bound Definitions
//*****>
newx = dex;
newy = dey;

enx = newx;
eny = dey + newEdgey;

MakeBars(newx, newy, stdz, enx, eny, stdz, size, r1, g1, b1, name);
//*****>
newx = dex + newEdgex;
newy = dey;

enx = newx;
eny = dey + newEdgey;

MakeBars(newx, newy, stdz, enx, eny, stdz, size, r2, g2, b2, name);

```



```

//*****>
//*****>Grid Lines
//*****>
    size = .2;

    newx = dex - 1;
    enx = dex + 1;

    newy = dey + newEdgey;
    eny = newy;
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, r1, g1, b1, name);
//*****>
    newy = dey + (.5 * newEdgey);
    eny = newy;
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, r1, g1, b1, name);
//*****>
    newy = dey + (.25 * newEdgey);
    eny = newy;
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, r1, g1, b1, name);
//*****>
    newy = dey + (.75 * newEdgey);
    eny = newy;
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, r1, g1, b1, name);
//*****>
    newx = dex + newEdgex - 1;
    enx = dex + newEdgex + 1;

    newy = eny = dey + newEdgey;
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, r2, g2, b2, name);
//*****>
    newy = eny = dey + (.5 * newEdgey);
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, r2, g2, b2, name);
//*****>
    newy = eny = dey + (.25 * newEdgey);
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, r2, g2, b2, name);
//*****>
    newy = eny = dey + (.75 * newEdgey);
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, r2, g2, b2, name);
//*****>
//*****>X-Axis definition
//*****>
    red = 1.0;
    green = 1.0;
    blue = 1.0;
    size = .5;

    newx = dex;
    newy = dey;

    enx = dex + newEdgex;
    eny = newy;
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, red, green, blue, name);
//*****>
    newy = dey - 1;

```

```

    eny = dey + 1;
    size = .2;
//*****>
    newx = enx = dex + (con11*(1+2*corr));
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, red, green, blue, name);
//*****>
    newx = enx = dex + (con12*(1+2*corr));
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, red, green, blue, name);
//*****>
    newx = enx = dex + (con21*(1+2*corr));
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, red, green, blue, name);
//*****>
    newx = enx = dex + (con22*(1+2*corr));
    MakeBars(newx, newy, stdz, enx, eny, stdz, size, red, green, blue, name);
//*****>
//*****>Turn by Turn Graphing
//*****>
    size = .25;
//*****>
    oldx = newx = dex;
    oldy = newy = dey;

    for (int ram = 0; ram < myround(edgex); ram++)
    {
        sum = rp[ram];
        sum /= turn;
        newy += sum * 100;
        newx += (ram * (1+(2*corr)));

        MakeBars(oldx, oldy, stdz, newx, newy, stdz, size, r1, g1, b1, name);

        oldx = newx;
        oldy = newy;
        newx = dex;
        newy = dey;
    }
//*****>
    oldx = newx = dex;
    oldy = newy = dey;
    sum = 0;

    for (int rom = 0; rom < myround(edgex); rom++)
    {
        sum = gp[rom];
        sum /= turn;
        newy += sum * 200;
        newx += (rom * (1+(2*corr)));
        MakeBars(oldx, oldy, stdz, newx, newy, stdz, size, r2, g2, b2, name);

        oldx = newx;
        oldy = newy;
        newx = dex;
        newy = dey;
    }

```

```

//*****>
//*****>Control Box Definitions
//*****>
    red = 1.0;
    green = 1.0;
    blue = 1.0;
    trans = 0.3;      //<- Degree of Transparency!!
//*****>
    double x1, x2, x3, x4, y1, y2, y3, y4;

    x1 = dex + con11*(1+2*corr);
    y1 = dey + newEdgey;

    x2 = dex + con12*(1+2*corr);
    y2 = dey + newEdgey;

    x3 = x2;
    y3 = dey;

    x4 = x1;
    y4 = dey;

    MakePlane(x1, y1, stdz, x2, y2, stdz, x3, y3, stdz, x4, y4, stdz, trans, red,
green, blue, other);
//*****>
    x1 = dex + con21*(1+2*corr);
    y1 = dey + newEdgey;

    x2 = dex + con22*(1+2*corr);
    y2 = dey + newEdgey;

    x3 = x2;
    y3 = dey;

    x4 = x1;
    y4 = dey;

    MakePlane(x1, y1, stdz, x2, y2, stdz, x3, y3, stdz, x4, y4, stdz, trans, red,
green, blue, other);
}

//*****>MYROUND FUNC-
TION<*****

int myround(double &val)
{
    int num;
    double dec;

    num = int(val);
    dec = val - num;

    if (dec == 0)
        return num;

```

```

else if (dec < 0){
    if (dec >= -.5)
        return num;
    else if (dec < -.5)
        return num-1;
}
else if (dec > 0){
    if (dec >= .5)
        return num+1;
    else if (dec < .5)
        return num;
}
cerr << "ERROR IN ROUNDING" << endl;
return 0;
}

//*****>MAKE-
SPHERES<*****

void MakeSpheres(double &ix, double &iy, double&iz, double &size, double &red,
double &green, double &blue, char file[50])
{
    ofstream outfile;
    outfile.open(file, ios::app);
    outfile << 2 << endl;
    outfile.setf(ios::scientific);
    outfile << ix << setw(15) << iy << setw(15) << iz << setw(12);
    outfile.unsetf(ios::scientific);
    outfile.setf(ios::fixed);
    outfile << size << setw(10) << red << setw(10) << green << setw(10) << blue <<
endl;
    outfile.close();
}

//*****>MAKE-
BARS<*****

void MakeBars(double &ix, double &iy, double &iz, double &kx, double &ky, dou-
ble &kz, double &rcyl, double &red, double &green, double &blue, char file[50])
{
    ofstream outfile;
    outfile.open(file, ios::app);

    outfile << 5 << endl;
    outfile.setf(ios::scientific);

    outfile
<< ix << setw(15) << iy << setw(15) << iz << setw(15) << rcyl << setw(15) << kx << setw(15) << ky <<
setw(15) << kz << setw(15) << rcyl;

    outfile.unsetf(ios::scientific);
    outfile.setf(ios::fixed);

    outfile << setw(12) << red << setw(10) << green << setw(10) << blue;

```

```

    outfile << endl;
    outfile.unsetf(ios::fixed);

    outfile.close();
}

//*****>SCRIPTER FUNCTION<*****
void scripter(int &select)
{
    char file[50], name[50], image[50], other[50];
    ofstream out;

    sprintf(file, "rscript");
    sprintf(other, "com.script");
    sprintf(name, "trac.r3d");
    sprintf(image, "image-%03d.tiff", select);

    out.open(file);
    out << "#!/bin/csh -f" << endl;
    out << "cat rotate.dat | ./rotate >! support.mat" << endl;
    out << "cat support.hda support.mat support.hdb graph.r3d " << name << "
con.r3d | render -tiff " << image << endl;
    out.close();

    /**Overlays labels with image**

    out.open(other);
    out << "#!/bin/csh -f" << endl;
    out << "convert -flip " << image << " " << image << endl;
    out << "combine -compose over " << image << " labels.tiff " << image << endl;
    out << "convert -enhance " << image << " " << image << endl;

    /**
    **

    out.close();

    system("chmod u=wrx rscript");
    system("chmod u=wrx com.script");
    system("rscript");
    system("com.script");
    //system("clear");
}

//*****>MOVER FUNCTION<*****
void mover(int &num)
{
    char command[200];

    for (int p = 1; p < num; p++)

```

```

    {
        sprintf(command, "mv image-%03d.tiff picts/", p);
        system(command);
    }

    system("rm rscript trac:r3d graph.r3d con.r3d com.script");
}

//*****>LINKED LIST REMOVE FUNCTION<*****
void remove(listType &list, int &id)
{
    nodePtr temp, marker, back;
    if (list.first == NULL)
        cout << endl;
    else{
        if (list.first->id == id){
            marker = list.first;

            if (list.first == list.last)
                list.first = list.last = NULL;
            else
                list.first = list.first->next;

            delete marker;
        }
        else {
            temp = list.first;

            while (temp != NULL && temp->id != id)
            {
                back = temp;
                temp = temp->next;
            }
            if (temp != NULL){
                marker = temp;
                back->next = temp->next;

                if (temp == list.last)
                    list.last = back;

                delete marker;
            }
        }
    }
}

```

DISTRIBUTION:

- 1 MS-0612 Review & Approval Desk, 4912
for DOE/OSTI
- 2 MS-0899 Technical Library, 4916
- 2 MS-1111 A. P. Thompson, 9225
- 2 MS-1111 G. S. Heffelfinger, 9225
- 1 MS-9018 Central Technical Files
- 2 MS-1674 T. A. Mehlhorn
- 1 MS-1231 W. M.. Knauf