

Novel Potentials for
the Simulation of Polyethylene
and other Polymeric Systems

by

George Waksman

Submitted to the Department of Materials
Science and Engineering in Partial
Fulfillment of the Requirements for the
Degree of

Bachelor of Science

at the

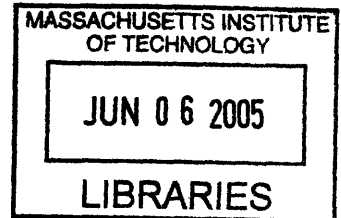
Massachusetts Institute of Technology

May 2005

[June 2005]

©2005 George Waksman

All rights reserved



The author hereby grants to MIT permission to reproduce and to
distribute publicly paper and electronic copies of this thesis document in whole or in part.

Signature of Author
Department of Materials Science and Engineering
May 13, 2005

Certified by
David K. Roylance
Associate Professor of Materials Engineering
Thesis Supervisor

Accepted by
Donald R. Sadoway
John F. Elliot Professor of Materials Chemistry
Chairman, Undergraduate Thesis Committee

Table of Contents

Table of Contents	2
Abstract	4
Theory	4
General	4
Order 0	6
Order 1	7
Order 2	8
Order 3	9
Extensions	9
Polypropylene, polybutylene, etc.....	10
Polyvinylchloride	10
Polyacetylene	10
Any Polymer	11
Implementation	11
General	11
Interactions.....	13
Order 0	13
Order 1	14
Order 2	15
Order 3	16
Temporal Integration	16
Temperature Regulation.....	17
Boundary Conditions	17
Initial Conditions	17
Results	18
Future Research	18
Appendix A: References	19
Appendix B: Equations	20
General	20
Variables	20
Common Equations.....	20
Order 0 Interactions	20
Interaction Potential	20
Order 1 Interactions	20
Interaction Potential	20
Order 2 Interactions	20
Interaction Potential	20
Constant Derivation	21
Order 3 Interactions	21
Interaction Potential	21
Constant Derivation	21
Appendix C: Constants	22
Order 0 Interactions	22
Order 1 Interactions	22

Order 2 Interactions	22
Order 3 Interactions	22
Appendix D: Figures.....	23
Interactions.....	23
Constant Derivations.....	24
Order 2 Interactions	24
Order 3 Interactions	25
In action	26
Appendix E: Implementation Code	39
Headers	39
Description.....	39
Code	39
makeoutfile	40
Description.....	40
Code	40
interact.....	48
Description.....	48
Code	48
atm2hr	56
Description.....	56
Code	56
atm2pov.....	57
Description.....	57
Code	57

Abstract

Throughout the history of science, people have been developing models to explain reality. The advent of computer technology has made it possible to devise and implement incredibly complicated numerical models in a relatively short period of time; for example, three-body problems, impossible to solve analytically, becomes trivial to model with computers. Beyond three-body problems, computers have been instrumental in solving many-body problems, such as those encountered in the atomic interactions within materials. Since computer modeling of atomic systems does not predate computers, it is still in its childhood, requiring further investigations. In order to further the development of computer modeling and a general understanding of reality, novel model algorithms for the simulation of polymeric systems have been developed. The proposed algorithms are empirical in nature, having been derived from observed atomic and molecular behavior, owing little to subatomic theories. The algorithms were developed to model polyethylene but extensions are provided to allow possible generalization to any other polymeric system.

Theory

General

The nature of molecular and atomic interactions is an incredibly complicated affair, currently best understood through the theories of quantum mechanics. Since quantum mechanics requires complex, continuous functions, which are very difficult to efficiently model on a computer, semi-empirical approximations of these interactions can be used in their place. Discrete element mechanics simulations of molecular and atomic interactions, often referred to as molecular dynamics simulations, can be conducted by

taking each atom to be a discrete simulation element. Thus taking each atom as a discrete element, the internuclear and electron interactions can be approximated using equations that are computationally simpler than the equations of quantum mechanics.

A common approximation used to simplify simulation is to remove the hydrogen atoms from the system, thereby greatly decreasing the simulation size. Different methods are used in order to account for the discrepancies introduced by the removal of the hydrogen atoms, such as changing the shape or arrangement of the carbon atoms to act as $\text{-CH}_2\text{-}$ structures. While this approximation saves computation time, it is not a part of the model presented in this paper as it makes for a less physically real system

In order to construct computationally reasonable approximations for the interactions that occur within polyethylene, it is necessary to break the interactions down into simple interactions between just a few atoms. These simple interactions, when combined in parallel serve as approximations for the entire system. In molecular dynamics simulations, the complexity and processing time increase at a greater and greater rate with regard to the number of atoms in a system when the number of atoms involved in a given interaction increases (computation goes as n^2 for two body interaction, n^3 for three body interactions and so on). Resulting from the complexity and processing time increases, the model presented in this paper has the requirement that all interactions be between no more than two atoms at once. In order to further keep interaction complexity down, interactions are constructed of polynomials of the least possible complexity.

The remaining consideration of how to break down the complex interactions within polyethylene into a number of simple interactions between pairs of atoms involves

the addition of the concept of interaction order. Interaction order, put simply, is defined by the number of covalent bonds that exist between two atoms. Thusly, two non-bonded atoms interact using order 0 interactions; two atoms directly bonded to each other interact using order 1 interactions; two atoms each bonded to the same atom but not each other interact using order 2 interactions and so on. Using this system of interaction order, algorithms for polyethylene simulation are presented with examples of systems for which a given interaction order is more characteristic. For the model presented in this paper atoms separated by more than 3 covalent bonds are considered infinitely far apart and interact using order 0 interactions.

Order 0

Order 0, or non-bonded, interactions in polyethylene need to account for steric interactions between hydrogen and carbon atoms. The hydrogen and carbon atoms within a polyethylene molecule are, roughly speaking, neutrally charged and thus interact only through induced dipole moments and electron shell repulsion. Since induced dipole moments and electron shell repulsion are the dominant forces in argon interactions, a system of argon gas is an ideal system to explore the order 0 interactions within polyethylene.

Induced dipole, also known as Van Der Waal's, interactions are attractive interactions that become stronger at shorter distances and diminish quickly as distance increases. Electron shell repulsion is a repulsive interaction that becomes incredibly strong at very short distances and diminishes to nearly zero almost immediately upon separation. These forms of interactions are well known and were modeled well before the advent of computers. One of the simplest, and most common, methods of simulating

these interactions is to use an algorithm based on the Lennard-Jones model for argon gas. Since the Lennard-Jones algorithms are based on argon gas models and 0 order interactions of polyethylene have been likened to the interactions of argon gas, we can simply change the constants and use the same algorithms. A Lennard-Jones 6-12 interaction is chosen because those in which the higher order term is the square of the lower order term decrease the number of necessary operations to obtain potentials.

Order 1

Order 1 interactions in polyethylene are the direct interactions between covalently bonded atoms. All of the bonds in polyethylene (hydrogen-carbon and carbon-carbon) are sigma bonds and are roughly equivalent to any other sigma bond in any other covalent material. In order to look at a simpler system when developing the algorithms for order 1 interactions in polyethylene, we will take a system in which interactions are dominated by order 0 and order 1 interactions. Gaseous, molecular hydrogen is a system dominated by order 1 interactions. The interactions between hydrogen molecules are essentially explained using the above order 0 interaction and thus the issue need not be belabored.

The sigma bond in molecular hydrogen is a bond that has been approximated rather accurately using quantum mechanics and molecular orbital theory. The approximated description of the sigma bond, so derived, is computational more complex than desired and so simplifications are sought. Since the strength of sigma bonds is substantially greater than the strength of Van Der Waal's interactions, if we impose the criteria that no bonds will be broken, we only need to approximate the sigma bond within a small region around equilibrium bond lengths. Near equilibrium, the potential of a sigma bond can be approximated quite accurately using a simple parabola. The use of a

parabola to represent a sigma bond is the same as considering a sigma bond to be a Hookean spring. The Hookean spring approximation for covalent bonds is fairly common and fits experimental results quite well.

Order 2

Order 2 interactions in polyethylene are those that arise from bond angle restrictions. Bond angle restrictions in polyethylene are very nearly identical to those in methane and arise from the sp^3 hybridization of a saturated carbon atom. Since the sp^3 hybridization in methane and polyethylene are nearly identical, we will investigate methane and then apply results to polyethylene.

Since we are trying to maintain bond angle, it would be intuitive to construct a three body potential and use cross products, bond lengths and the law of cosines to determine the angle and then write a potential based on the calculated angle. A simple but effective potential based on bond angle, would be one that is parabolic around the equilibrium angle. Parabolic potentials with regard to bond angle have been used in the past and backed up with experimental information.

Sadly, intuition leads us to a potential that does not satisfy our design requirement for purely polynomial two body potentials. In order to satisfy the design requirements a simpler potential was developed. If we make the assumption that all order 1 bonds will be near equilibrium, single bonds are of similar length and use the small angle approximation for sine, we can derive an equation which relates distance between atoms to bond angle. Having a relation between distance and bond angle, we can take a potential that is parabolic in bond angle and construct a simple potential in atomic

distance for two atoms. The Hookean spring was chosen to serve as the order 2 interaction potential.

Order 3

Order 3 interactions in polyethylene exist to account for torsional energy. In simpler simulations it is likely acceptable to ignore torsional energy terms as steric hindrance will minimize carbon-carbon eclipsing effects, however for completeness, we will address torsional effects to prevent hydrogen-hydrogen eclipsing and more accurately describe carbon-carbon eclipsing. Since torsional effects are first (by number of carbons) observed in ethane, we will take it to be our model.

In much the same way as with our order 2 interaction, a relation between bond angle and atomic distance is derived to prevent a need for many-body potentials. Unlike, the order 2 interaction, we can not use a parabolic potential or we will prevent torsion altogether. Instead, a repulsive potential is used to discourage eclipsed conformations. By maximizing repulsion at times when atoms are eclipsed and minimizing repulsion when they are staggered, the desired torsional effects are accomplished

Extensions

Through the modification of some of the parameters and constants used in the model of polyethylene it is possible to simulate other polymeric systems. With the further addition of a few new interactions it becomes theoretically possible to simulate any polymeric system. Such possible extensions on the simulation of polyethylene are presented here.

Polypropylene, polybutylene, etc.

Polypropylene, polybutylene, polyisopropylethylene and any other saturated hydrocarbon polymer may be easily simulated using the proposed algorithms without modification. Some modification of constants might be necessary to accommodate changes in order 2 interactions around carbons bonded to more than two other carbons but the algorithms should need no alteration.

Polyvinylchloride

In polyvinylchloride, the assumption that all species are neutrally charged is no longer valid. The chlorine atom, being more electronegative, draws electron density away from the rest of the molecule. In order to account for the differences in electron density, it would be necessary to introduce a new order 0 interaction. The interaction, meant to deal with charge differences, could be a simple, parabolic, Coulombic potential. Every species within the system would need to be assigned a partial charge, adding another set of constants to the simulation.

Polyacetylene

Polyacetylene being a stereotypical, conjugated, unsaturated hydrocarbon introduces the difficulty of multiple bonds. Adding multiple bonds changes the nature of the order 1, order 2 and order 3 interactions in the system. The order 1 and order 2 interaction changes are trivial and simply require the adjustment of constants.

The order 3 interaction necessary to simulate polyacetylene is not trivial and must be created to prevent torsion around the double bond. It may be the case that a narrow parabola, as was avoided in the polyethylene order 3 interaction, could achieve the desired prevention of torsion but it might overwhelm other interactions within the system and the matter requires further investigation.

That polyacetylene is a conjugated polymer introduces yet another order 3 difficulty. If the polyacetylene is described using alternating single and double bonds, the single bonds will allow free-rotation, which does not happen in a conjugated system. This problem may be solvable by treating single bonds (or all carbon-carbon bonds) as double bonds and adjusting constants accordingly.

Any Polymer

Using the base algorithms and the suggested extensions, it should be possible to simulate any polymeric system and some covalent and ionic systems. All that will be required is adjustment of various constants and initial conditions. The addition of solvents is easily accomplished by treating solvents in the same manner as polymers.

The inclusion of interacting species is outside the scope of these algorithms and would require more complex extensions than those provide.

The presented algorithms for polyethylene are meant to be viewed both for their value in simulating saturated hydrocarbons and for their possible use as the groundwork for more complex polymer simulation.

Implementation

General

New software was developed specifically for the purposes of implementing the newly devised algorithms. The software was written using the C programming language in a manner meant to be compatible with all current operating systems. Software was developed to provide output from the simulations in both human readable formats and in a format that could be used by the free software tool POV-Ray to generate still images. The developed software was fairly slim in features and power, designed to serve as a

platform for testing the algorithms and not to replace pre-existing molecular dynamics packages. After the software was developed, the algorithms were implemented in a bottom up manner; implementing interactions one order at a time and then moving on.

Due to time constraints, correlation between experimental results and simulated results could not be conducted in any quantitative manner. Simulation was compared to theory and experiment on a qualitative basis. As a result of the lack of quantitative comparison many of the constants used are drawn from literature or have been chosen for simulation specific reasons. If further research is to be conducted, an important step will be comparison to experimental results and modification of simulation constants. Since this model deals with interactions up to order 3, a good place to start would be with a comparison between simulated and experimental data with regards to ethane.

Design and selection of constants was conducted first by drawing straight from literature when possible. When not possible to draw straight from literature, some constants were drawn from literature and theoretically modified. Other constants were intuited using similar values from literature or rough ideas of what was going on. After constants were initially devised, some were modified and tweaked in order to attain values that would prevent the system from “blowing up”. If a constant is of a sufficiently inappropriate value, in some cases atoms would be forced into positions of incredibly high potential, from which they would be shot out at very high forces cause the system to “blow up”

The test implementation’s source code is provided in Appendix E: Implementation Code.

Interactions

Order 0

Order 0 interactions are both the easiest to implement and the most computational intensive. With more complicated implementation schemes, computational power can be saved but, due to the test platform nature of the software system, order 0 interactions became the limiting factor in simulation size. Each atom is interacted with each other atom once per time step, resulting in a number of pair-wise interactions approximately equal to the number of atoms in the system squared per time step.

Neighbor Lists

In order to speed up calculations, a system of neighbor lists was implemented. In a neighbor list system all atoms near a given atom are recorded in a list. Instead of interacting said given atom with all other atoms, it is interacted only with those atoms on its list. To generate the neighbor list one must compare every possible pair of atoms to see if they are within a specific cutoff distance of each other, which still takes a substantial amount of time but, since this calculation can be done every few time steps, time is saved overall. The neighbor list method for optimizing order 0 interactions is a very simple one and more complex ones could be used, such as by partitioning space into a binary tree or by any of many methods commonly used in discrete element simulations.

Constants

The Lennard-Jones equation has two constants; σ and ϵ . σ represents the lowest energy distance between the two atoms. ϵ represents the depth of the potential well between the two atoms; the difference in potential between minimum energy and infinite distance.

σ

σ is taken to be the sum of the radii of the two atoms as taken from literature.

ϵ

ϵ is taken to be the square root of the product of the ϵ values for the two interacting atom types. Values for ϵ for hydrogen and carbon could not be located in literature and were devised based on values of ϵ for other materials and then adjusted to allow the system to work well. ϵ values were taken to be similar to those used for argon simulation in literature. The nature of the method for choosing the value for ϵ is not perfect and further research should be conducted to obtain better values. A side effect of the uncertainty of the value of ϵ is that the intermolecular and entropic forces (strongly influenced by order 0 interactions) will not be entirely accurate.

Order 1

Order 1 interactions were implemented by setting atomic bonds during the initial set up and then maintaining a list of bonded pairs. These bonded pairs were then interacted every time step.

Constants

Hookean spring systems have two constants, x_0 and k . x_0 represents the equilibrium distance between spring ends; equilibrium bond length in our system. k —represented as k_0 to prevent confusion with the often used counter variable k in the software code—represents the stiffness of the spring.

x_0

x_0 was taken to be the equilibrium bond lengths for carbon-carbon and carbon-hydrogen bonds in ethane from literature.

k0

k_0 was taken to be the bond stiffness for carbon-carbon and carbon-hydrogen bonds in ethane from literature.

Order 2

Order 2 interactions were implemented by traversing the atomic bonds set aside for order 1 interactions to find pairs of atoms separated by one intermediary atom. These pairs were then interacted

Constants

Since order 2 interactions are modeled as Hookean springs, they have the same constants as the order 1 interactions.

x0

Using the equilibrium bond lengths for single bonds from literature, the equilibrium bond angles from literature and the law of cosines, values for x_0 were calculated.

k0

Since k_0 is only indirectly related to bond angle stiffness and has been devised to serve an approximation unique to these algorithms a value had to be intuited and worked out through trial and error. The value for k_0 had to be large enough to maintain a fairly tetrahedral arrangement around a carbon atom and small enough not to heavily alter the order 1 interactions. Due to the use of angles and the law of cosines, the affect of changing one of the single bonds does not affect the order 2 distance much, minimizing back effects of the order 2 interaction on order 1 constants.

Order 3

Order 3 interactions were implemented in much the same way as order 2 interactions, by traversing order 1 bonds.

Constants

The order 3 interactions use two constants x_0 and z . The order 3 x_0 is not like the x_0 of order 1 or 2; the order 3 x_0 represents the distance between two atoms when they are in a completely eclipsed configuration. The constant z is a scalar variable that represents the energy change from the completely eclipsed position and the staggered position. Both of these constants have been derived geometrically from experimental values.

x_0

Using equilibrium bond lengths, equilibrium bond angles and geometry, values for x_0 were calculated.

z

By calculating the distance between atoms in the staggered position using equilibrium bond lengths, equilibrium bond angles, geometry and combining with values for x_0 and experimental values for the change in energy, values for z were calculated

Temporal Integration

The Velocity Verlet algorithm was chosen as the method for time integration for a number of reasons. The Velocity Verlet algorithm is a fairly simple algorithm that is not very computationally intensive and is fairly stable. A further advantage to using the Velocity Verlet algorithm is that it is a conservative algorithm, neither introducing nor destroying energy in its integrations.

Time steps were conducted at a rate of one step per femtosecond. This choice was made because atomic bond oscillations are on the order of femtoseconds and early simulation runs had a strong tendency to “blow up” when time steps were performed less often.

Temperature Regulation

In order to deal with minor program limitations and minimize a number of minor problems in the implementation, an algorithm was devised to maintain a roughly constant system temperature. The temperature was calculated during every time step and the velocity of every atom in the system was adjusted by the square root of the ratio of temperature and desired temperature. Through this algorithm, conservation of system energy was lost but it made the simulations substantially more stable and helped to eliminate accidental sources of energy imparted by the initial conditions.

Boundary Conditions

Periodic boundary conditions were chosen because of their ease of implementation and physical simplicity. By using periodic boundaries, the issue of what to do when an atom approaches or crosses a boundary becomes trivial; the atom simply appears on the opposite side of the system. Since the system becomes periodic, it can be considered to repeat ad infinitum, theoretically representing a uniform bulk system, with measurable local properties.

Initial Conditions

The issue of initial conditions is a very complicated one and can have profound effects on the results of simulations. Because periodic arrangements are easiest to enumerate, systems consisting of long stretched chains of polyethylene were designed. In

these systems, each atom was given a random initial velocity vector and the system velocities were normalized to specified temperature.

Results

Although not quantitative, very promising qualitative results have been obtained. Bond lengths were maintained, though some variation occurred as energy was stored in bond oscillations. Tetrahedral arrangements around carbon atoms were maintained, with some variance also due to oscillatory energy storage. Entropic chain contractions were observed. Condensation of long chains was seen to take place very rapidly. For more qualitative results, see Appendix D: Figures.

Future Research

There are two primary areas for future research with regards to the model and algorithms presented in this paper. The first is improvements to the model and the second is improvements to the implementation.

There are a number of possible improvements to the model. A first such improvement is to correlate the constants used in the equations with experimental data. A second would be to devise and implement some of the extensions suggested for simulating other polymers.

As for improvements to the implementation, there are many, primarily centered on improving speed and system size. Improvements could be made to the individual algorithms for each interaction. The software could be cleared of unnecessary computations. The system could be broken down spatially as a replacement for neighbor lists. By far the best improvement would be seen by making the software highly parallel and using multiple computers/multi-processor computers.

Appendix A: References

- Lennard-Jones, J. E. On the Determination of Molecular Fields. II. From the Equation of State of a Gas. Proceedings of the Royal Society of London, Series A, 1924, v106, p463.
- Rahman, A. Correlations in the Motion of Liquid Argon. Physical Review, 1964, v136, pA405.
- Verlet, L. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. Physical Review, 1967, v159, p98.
- Padding, J. T.; Briels, W. J. Uncrossability Constraints in Mesoscopic Polymer Melt Simulations: Non-Rouse Behavior of C₁₂₀H₂₄₂. Journal of Chemical Physics, 2001, v115, p2846.
- Halley, J. W.; Duan, Y.; Nielsen, B. Simulation of Polyethylene Oxide: Improved Structure Using Better Models for Hydrogen and Flexible Walls. Journal of Chemical Physics, 2001, v115, p3957.
- Lin, B.; Boinske, P. T.; Halley, J.W. A Molecular Dynamics Model of the Amorphous Regions of Polyethylene Oxide. Journal of Chemical Physics, 1996, v105, p1668.
- Guo, H. X.; Yang, X. Z.; Li, T. Molecular Dynamics Study of the Behavior of a Single Long Chain of Polyethylene on a Solid Surface. Physical Review E, 2000, v61, p4185.
- Weber, T. A.; Helfand, E. Molecular Dynamics Simulation of Polymers. I. Structure. Journal of Chemical Physics, 1979, v71, p4760.
- Krishna Pant, P. V.; Han, J.; Smith, G. D.; Boyd, R. H. A Molecular Dynamics Simulation of Polyethylene. Journal of Chemical Physics, 1993, v99, p597.
- Bharadwaj, R. K.; Boyd, R. H. Effect of Pressure of Conformational Dynamics in Polyethylene: A Molecular Dynamics Simulation Study. Macromolecules, 2000, v33, p5897.
- Lide, D. R. Editor. CRC Handbook of Chemistry and Physics, 80th Edition. CRC Press, 1999.
- Wade, L. G. Organic Chemistry, 4th Edition. Prentice Hall, 1999.

Appendix B: Equations

General

Variables

These variables are used throughout Appendix B

\vec{r} = displacement from one atom to another

r = distance from one atom to another

U = potential energy generated between two atoms

\vec{f} = force exerted on one atom by the potential field

Common Equations

These equations will be used throughout Appendix B

$$\vec{f} = -\nabla U$$

$$\vec{f}_1 = -\vec{f}_2$$

Order 0 Interactions

Interaction Potential

$$U = -4\varepsilon \left(\frac{\sigma^6}{r^6} - \frac{\sigma^{12}}{r^{12}} \right)$$

$$\vec{f} = -24 \frac{\varepsilon}{r^2} \left(\frac{\sigma^6}{r^6} - 2 \frac{\sigma^{12}}{r^{12}} \right) \vec{r}$$

Order 1 Interactions

Interaction Potential

$$U = \frac{1}{2} k_0 (r - x_0)^2$$

$$\vec{f} = -k_0 \left(\frac{r - x_0}{r} \right) \vec{r}$$

Order 2 Interactions

Interaction Potential

$$U = \frac{1}{2} k_0 (r - x_0)^2$$

$$\vec{f} = -k_0 \left(\frac{r - x_0}{r} \right) \vec{r}$$

Constant Derivation

The variables and geometries used for derivation of constants are in Appendix D: Figures

$$x_0 = \sqrt{a^2 + b^2 - 2ab \cos(C)}$$

Order 3 Interactions

Interaction Potential

$$U = z \frac{x_0^2}{r^2}$$

$$\bar{f} = 2z \frac{x_0^2}{r^4} \bar{r}$$

Constant Derivation

The variables and geometries used for derivation of constants are in Appendix D: Figures

$$l = a \cos(A) + b + c \cos(C)$$

$$x_0 = \sqrt{l^2 + (a \sin(A) - c \sin(C))^2}$$

$$f_1 = a^2(1 - \cos(F_1))$$

$$f_2 = c^2(1 - \cos(F_2))$$

$$P = \sqrt{x_0^2 + \left(\frac{f_1 + f_2}{2}\right)^2} - x_0$$

$$z = \frac{\Delta U}{2 \left(1 - \frac{x_0^2}{(x_0 + P)^2}\right)}$$

Appendix C: Constants

Order 0 Interactions

Atom Type	Sigma (Å)	Epsilon (J)
Argon (for comparison)	3.4	1.66e-21
Carbon	0.77	1.00e-21
Hydrogen	0.32	1.00e-21

Order 1 Interactions

Bond	x0 (Å)	k0 (N/m)
C-H	1.12	483
C-C	1.53	450

Order 2 Interactions

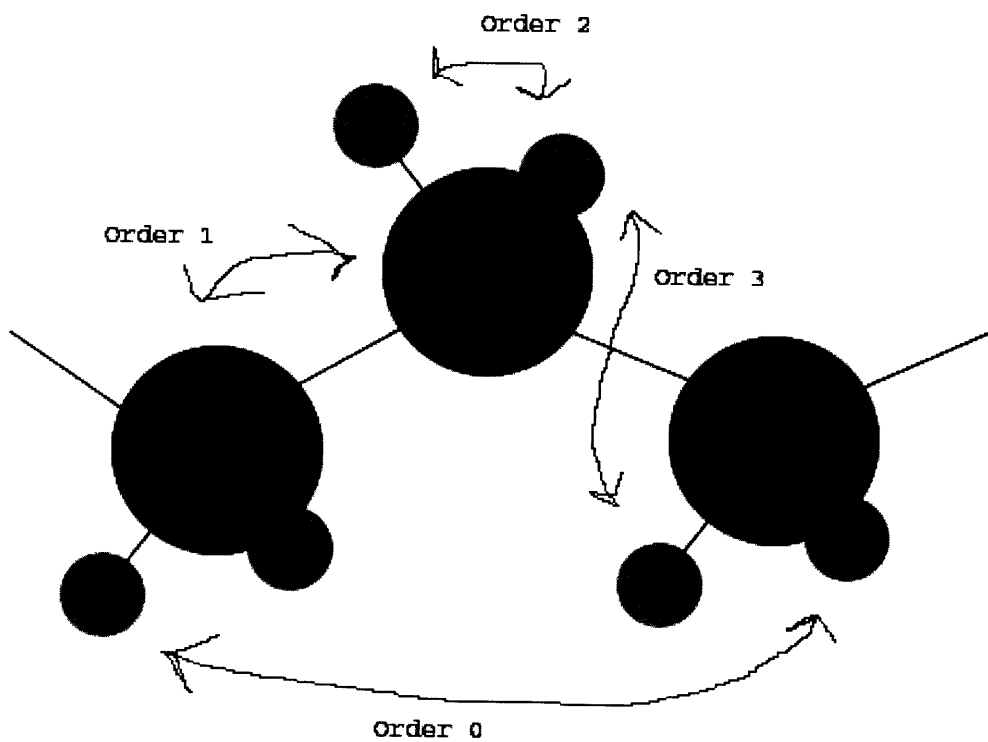
Angle Type	x0 (Å)	k0 (N/m)
H-C-H	1.83	10
H-C-C	2.18	10
C-C-C	2.50	10

Order 3 Interactions

Interaction Type	x0 (Å)	z (J)
H-C-C-H	2.278	1.196e-19
H-C-C-C	2.446	1.117e-19
C-C-C-C	2.551	1.330e-19

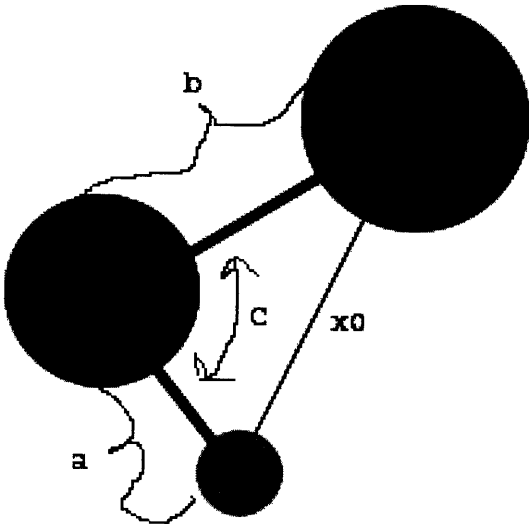
Appendix D: Figures

Interactions

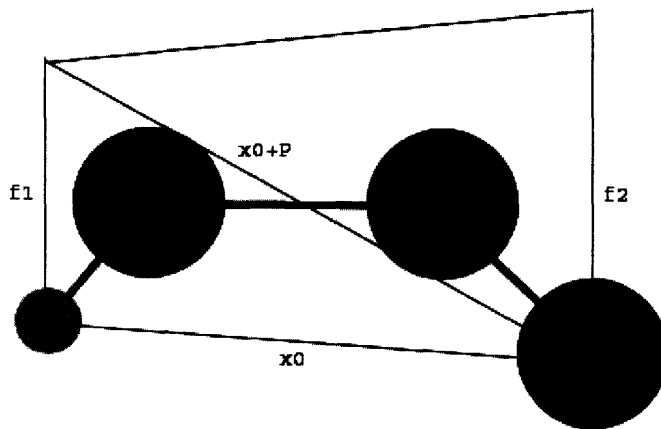
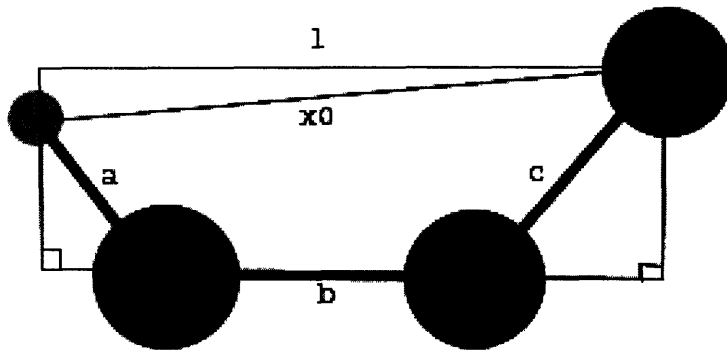


Constant Derivations

Order 2 Interactions

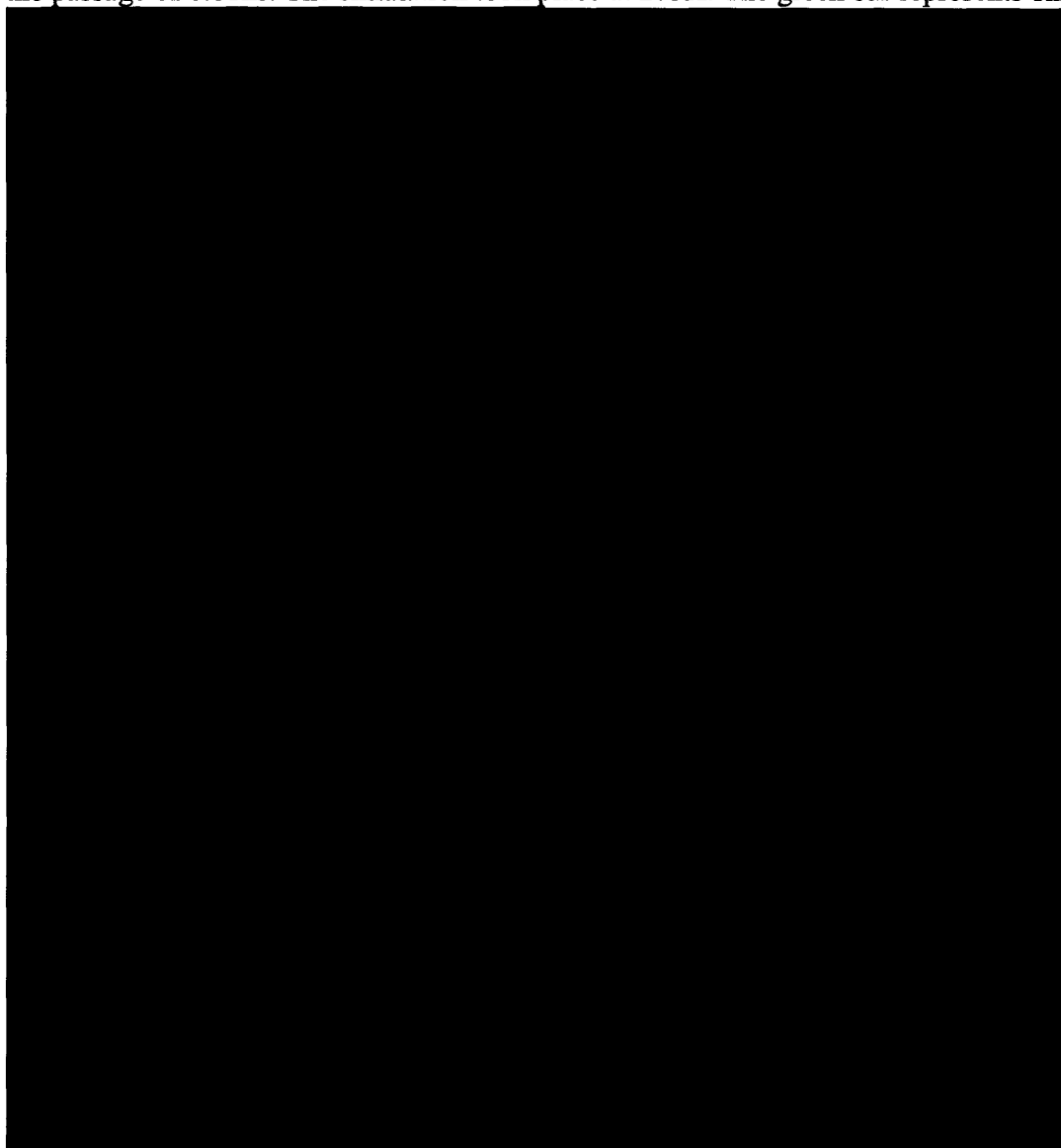


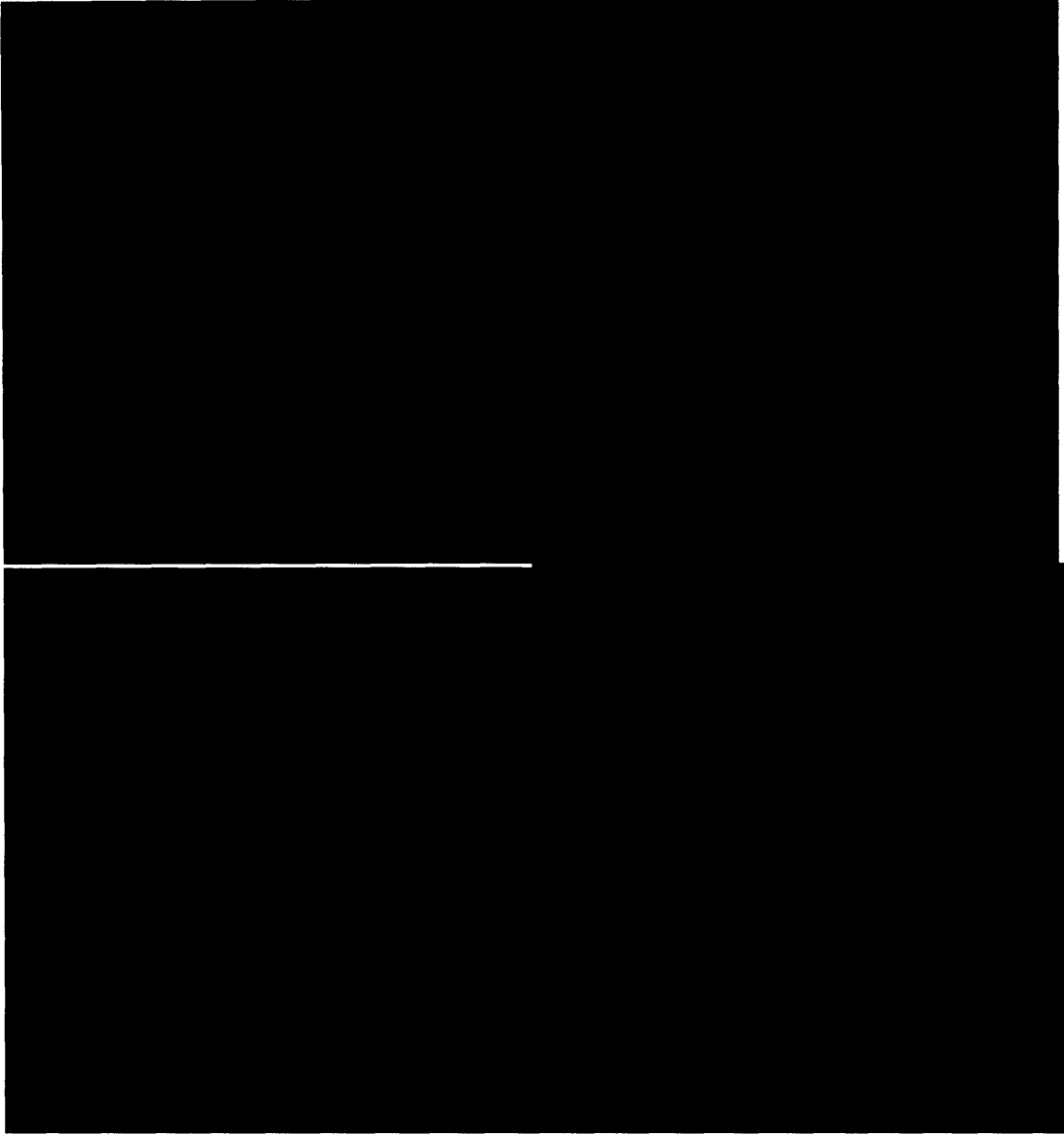
Order 3 Interactions



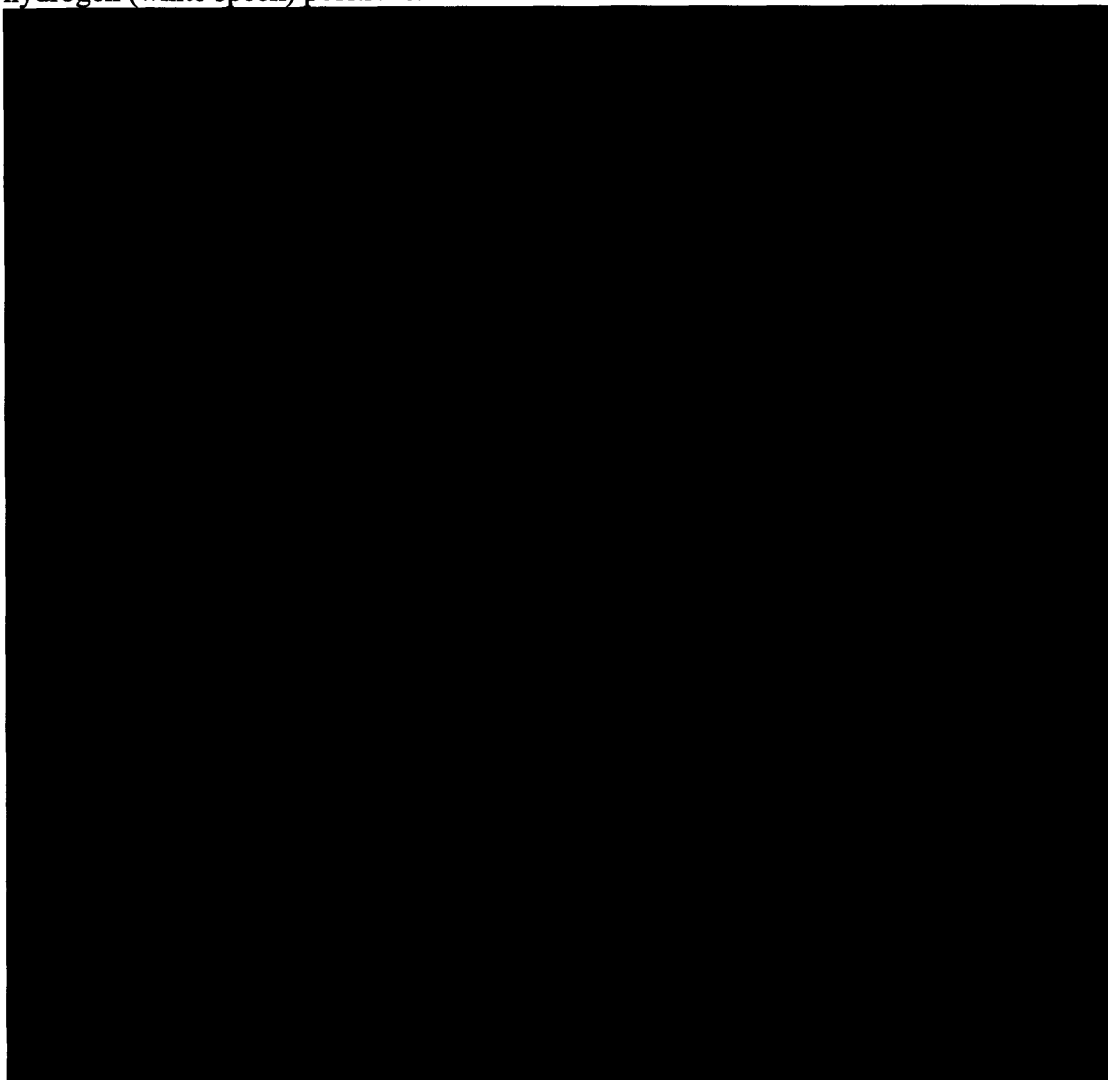
In action

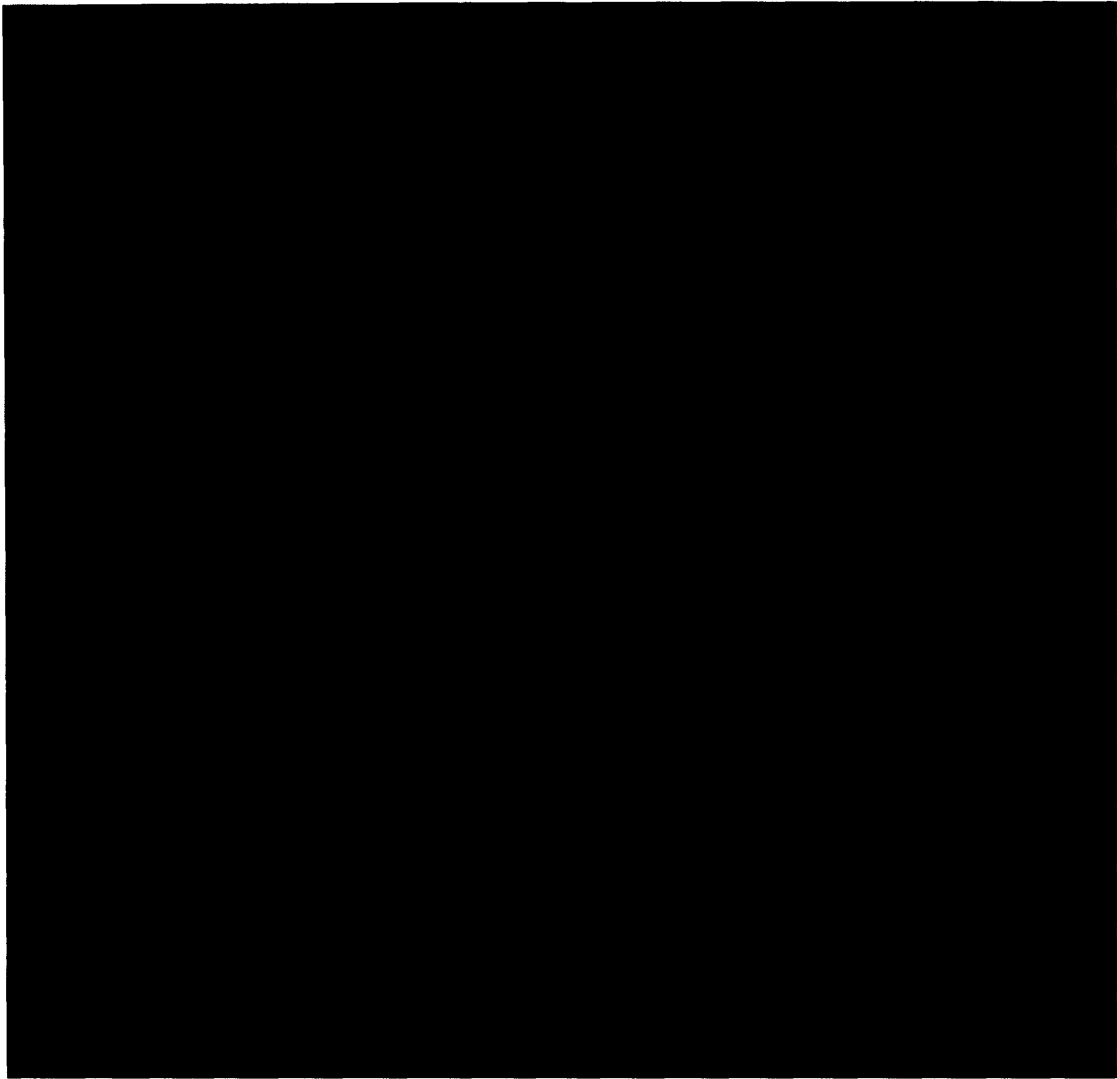
Here is a single chain of $C_{200}H_{402}$ starting from a stretched state. Each image represents the passage of 0.01ns. The simulation took place at 273K. The green bar represents 1nm.

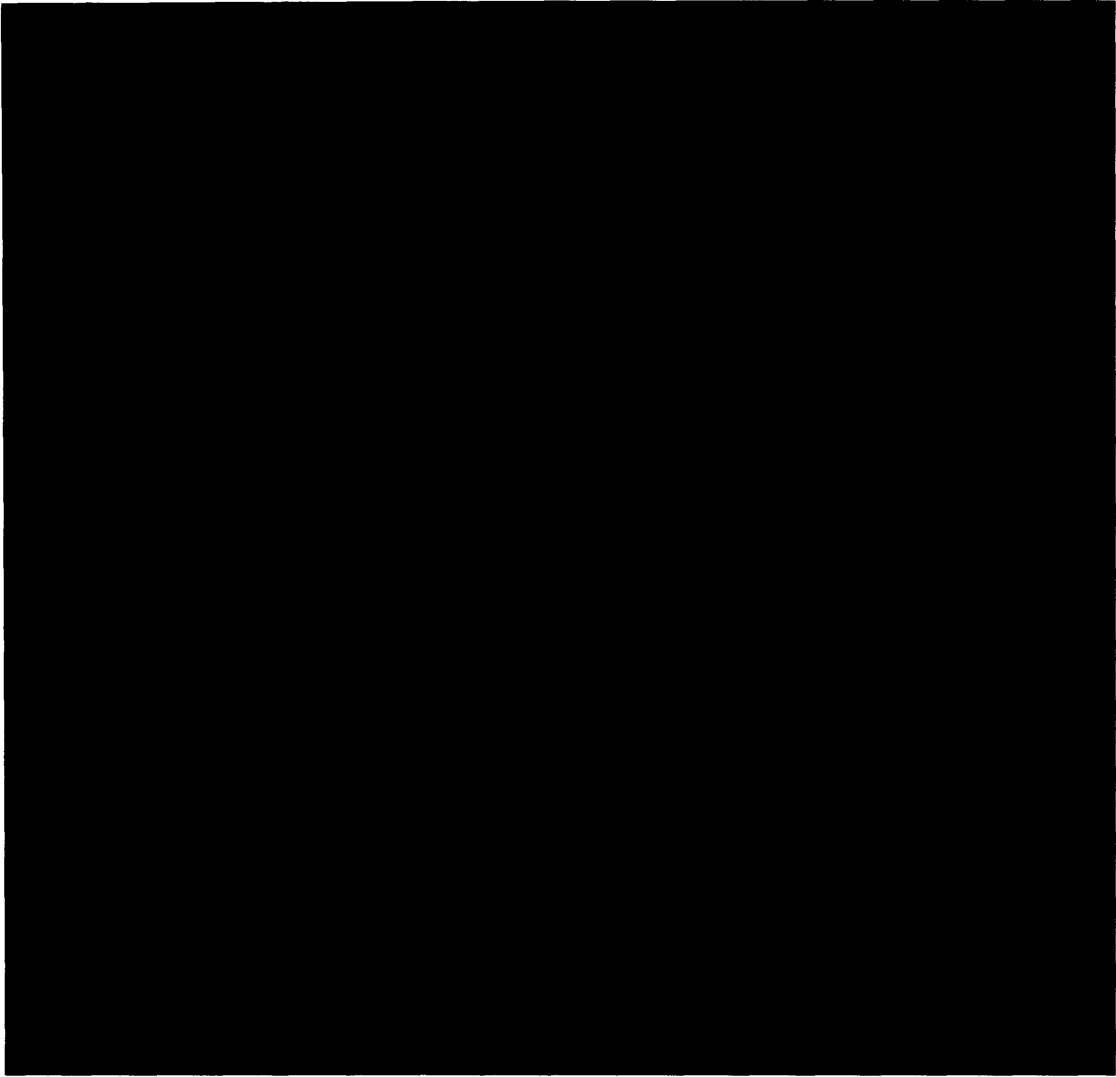




Here is a single chain of $C_{4000}H_{8002}$ interacting at 350K, starting, again, from a stretched state. Here every frame is 0.1ns. The green bar again represents 1nm. The color contrasts and atom sizes may make it difficult to make out but some can be inferred from the hydrogen (white speck) positions.

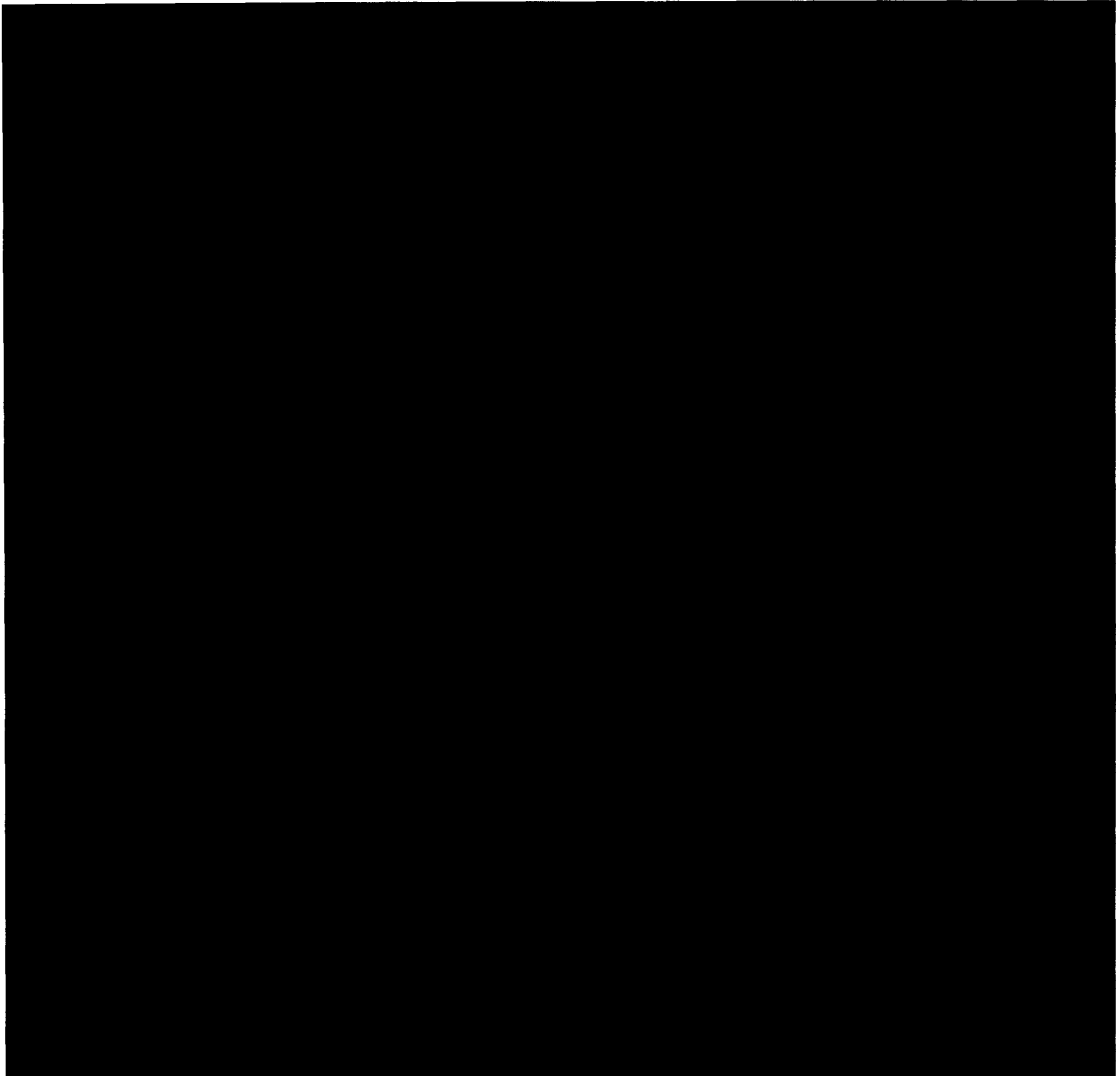




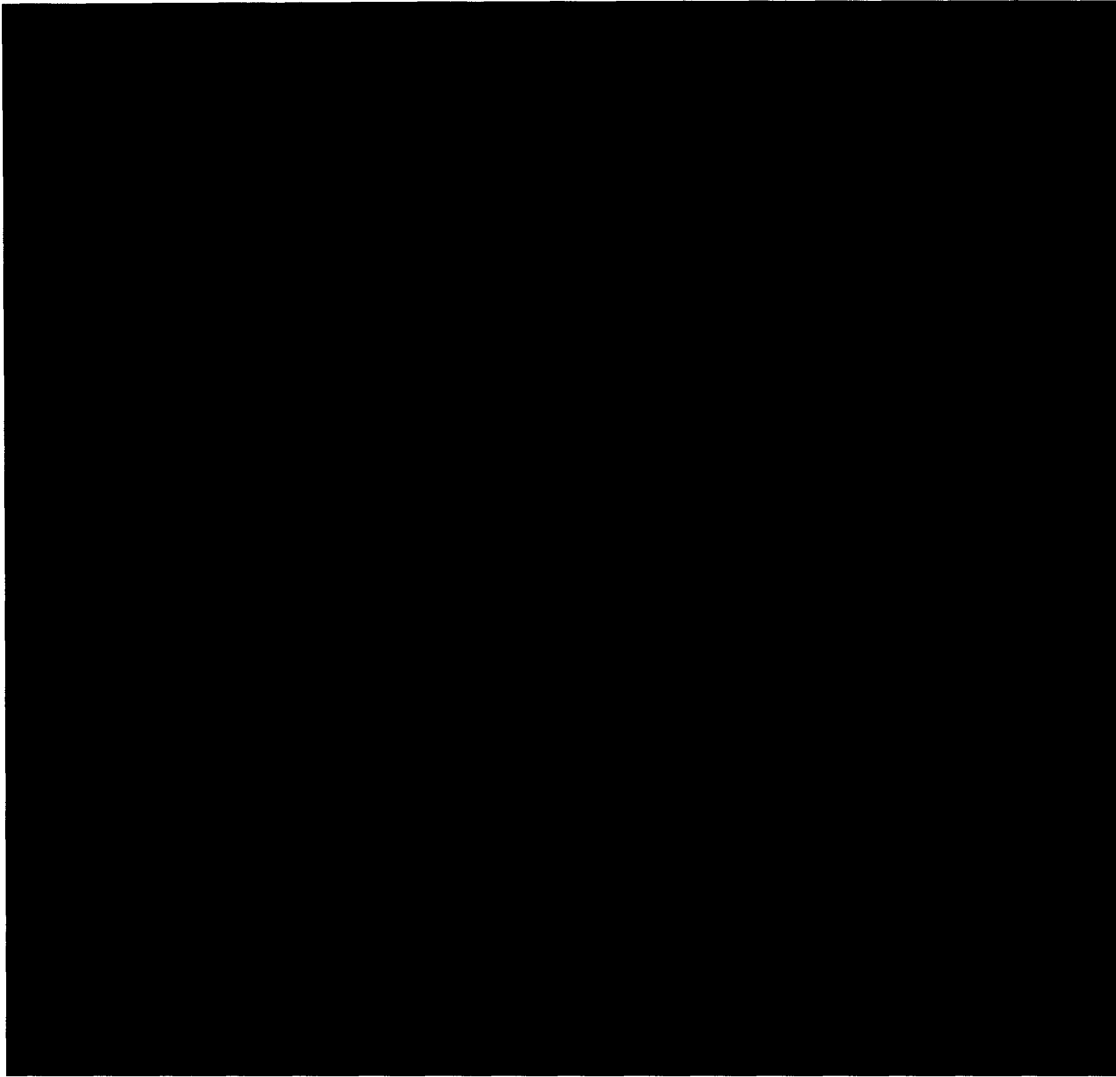


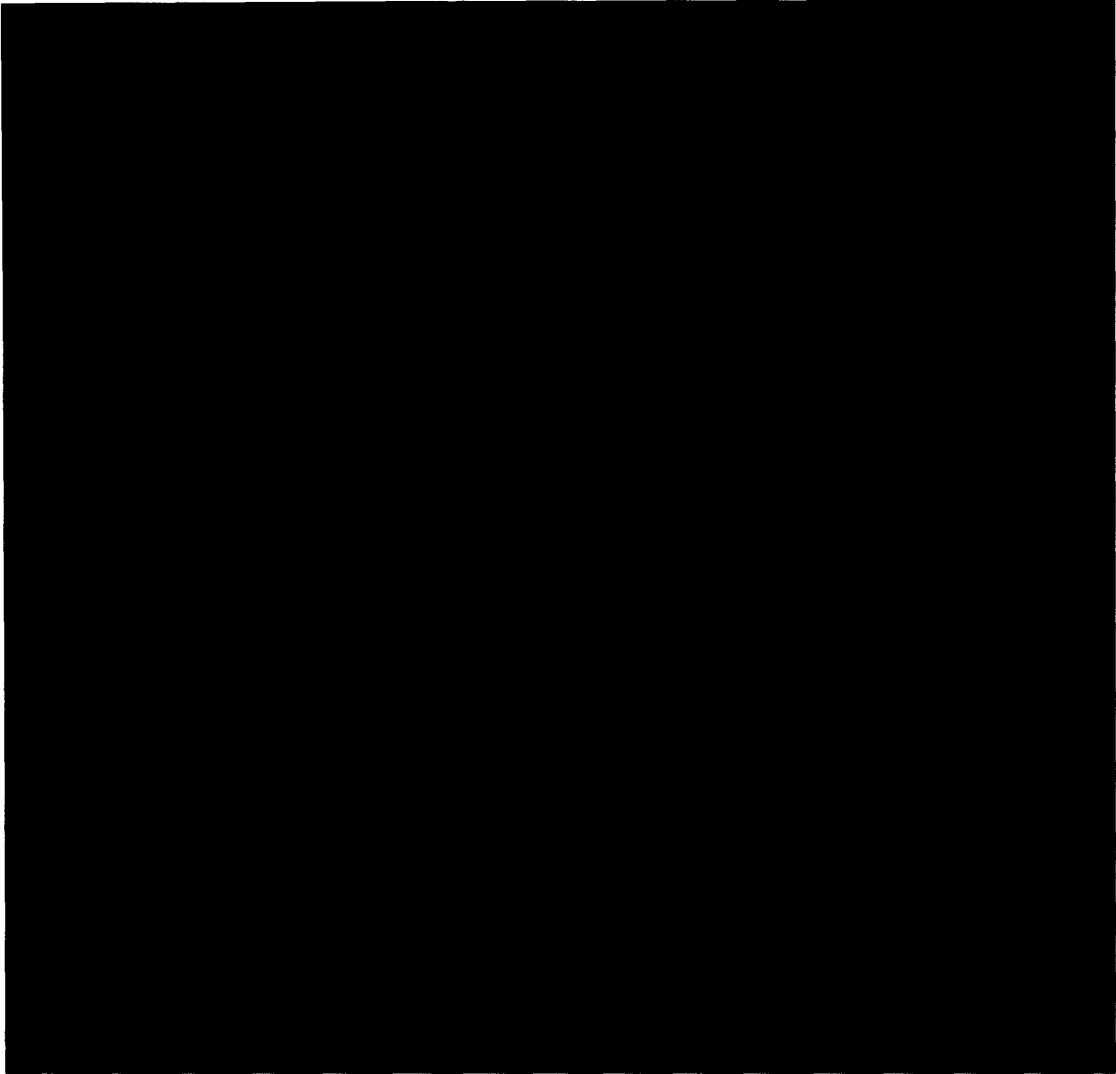


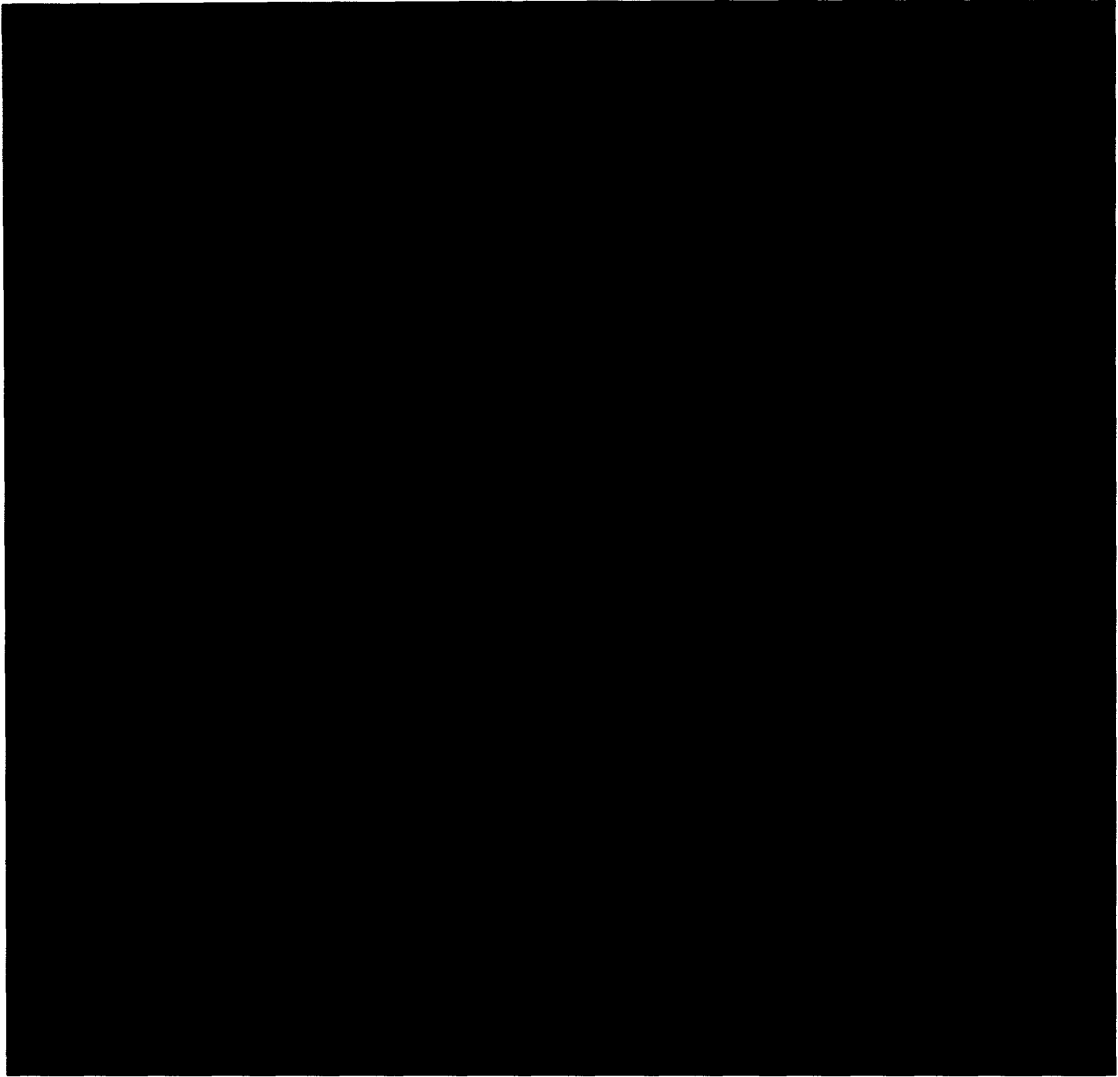


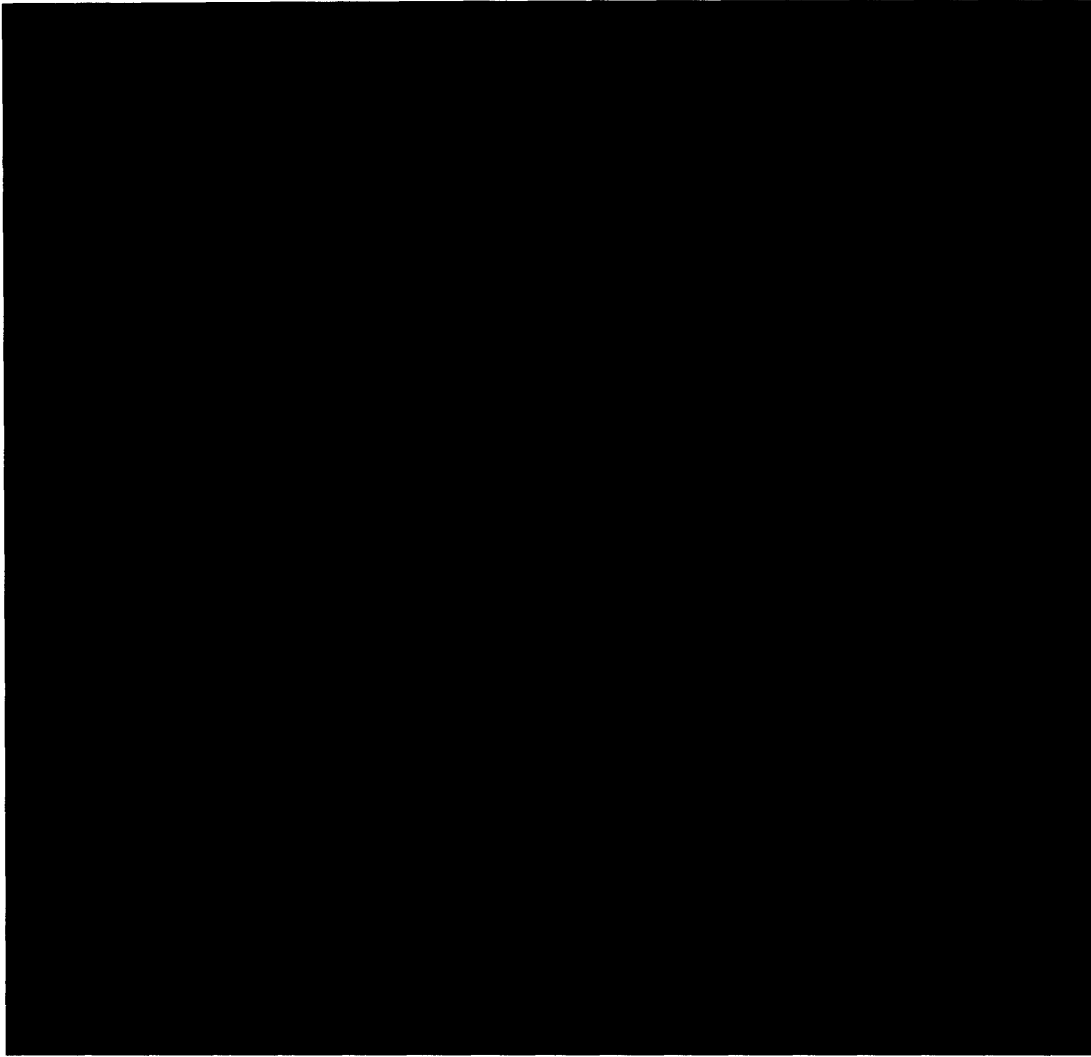












Appendix E: Implementation Code

An apology must be made for breaks in lines of code that have been introduced by their inclusion in a formatted document

Headers

Description

There are a number of files containing information and data structures that must be common to all of the various applications. These are put separately in C header files for convenience.

Code

atoms.h

```
#ifndef _atoms_h_
#define _atoms_h_

#include "types.h"

#define NUMBONDS 4

typedef struct
{
    enum AtomTypes type;
    double x,y,z;
    double vx,vy,vz;
    double ax,ay,az;
    double potential,newpotential;
    double ndx,ndy,ndz;

    int bonds[NUMBONDS];
} Atom;

#endif
```

header.h

```
#ifndef _header_h_
#define _header_h_
typedef struct
{
    int iterations;
    double mincutoff;
    double dt;
    double bigX,bigY,bigZ;
    double desiredT;
    double T;
    int numatoms;
} HeaderInfo;

#endif
```

types.h

```
#ifndef _types_h_
#define _types_h_

typedef struct
{
    const char* symbol;
    double epsilon;
    double radius;
    double mass;

    const char* pigment;
}
```

```

} AtomType;

enum AtomTypes {fake, C, H};
const AtomType elements[] = {
    {"fake", 1.0, 0.2, 0.1, "White"},
    {"C", 1.0E-21, 0.77E-10, 1.994E-26, "Gray20"},
    {"H", 1.0E-21, 0.32E-10, 1.673E-27, "White"}
};

#endif

```

makeoutfile

Description

The **makeoutfile** application is a program that creates an input file to be used by the **interact** application. It will create a system containing a number of strands of polyethylene at 273K.

Code

makeoutfile.h

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "atoms.h"
#include "header.h"

#define XMUL 1
#define YMUL 1
#define ZMUL 1
#define N 2000

#define TEMP 350.0

const double sqrt2 = 1.4142135623730950488016887242097;
const double sqrt3 = 1.7320508075688772935274463415059;
const double kB = 1.380658E-23;
const double RMo2 = (double)RAND_MAX / 2.0;

int main()
{
    Headerinfo tempheader;
    Atom* allatoms;
    int currentatom;
    int numatoms;
    FILE* file;
    int i,j,k,l,n;
    double currentT;
    double vMul;
    double netmx,netmy,netmz;
    double a;

    file = fopen("infile.atm", "wb");
    if(file == NULL)
    {
        printf("Failed to open infile.atm for writing\n\n");
        exit(6);
    }

    numatoms = (2+6*N)*XMUL*YMUL*ZMUL;

    allatoms = (Atom*)malloc(numatoms*sizeof(Atom));
    if(allatoms == NULL)
    {
        printf("unable to allocate memory for allatoms\n\n");
        fclose(file);
        exit(5);
    }

    /*srand((unsigned int)time(NULL));*/
    srand(2197);

```



```

    a = (elements[(enum AtomTypes)H].radius + elements[(enum AtomTypes)C].radius) *
2.0 / sqrt3;

    netmx = netmy = netmz = 0.0;
    currentT = 0.0;
    tempheader.iterations = 1000000;
    tempheader.mincutoff = 10.0 * a;
    tempheader.dt = 1.0E-15;
    tempheader.bigX = a + 3.2 * (double)XMUL * a + a*(double)N;
    tempheader.bigY = a + 2.4 * (double)YMUL * a + a*(double)N;
    tempheader.bigZ = a + 3.8 * (double)ZMUL * a + a*(double)N;
    tempheader.T = TEMP;
    tempheader.desiredT = TEMP;
    tempheader.numatoms = numatoms;

    currentatom = 0;
    for(i=0; i<XMUL; i++)
    {
        for(j=0; j<YMUL; j++)
        {
            for(k=0; k<ZMUL; k++)
            {
                allatoms[currentatom].type = C;
                allatoms[currentatom].x = 2.0 * a + 3.2 * (double)i * a;
                allatoms[currentatom].y = 2.0 * a + 2.4 * (double)j * a;
                allatoms[currentatom].z = 2.0 * a + 3.8 * (double)k * a +
0.25 * a * (double)N;
                allatoms[currentatom].vx = ((double)rand()-RMO2)/RMO2;
                allatoms[currentatom].vy = ((double)rand()-RMO2)/RMO2;
                allatoms[currentatom].vz = ((double)rand()-RMO2)/RMO2;
                allatoms[currentatom].ax = 0.0;
                allatoms[currentatom].ay = 0.0;
                allatoms[currentatom].az = 0.0;
                allatoms[currentatom].ndx = 0.0;
                allatoms[currentatom].ndy = 0.0;
                allatoms[currentatom].ndz = 0.0;
                allatoms[currentatom].potential = 0.0;
                allatoms[currentatom].newpotential = 0.0;
                for(l=0; l<NUMBONDS; l++) allatoms[currentatom].bonds[l] =
0;
                allatoms[currentatom].bonds[0] = 1;
                allatoms[currentatom].bonds[1] = 2;
                allatoms[currentatom].bonds[2] = 3;
                allatoms[currentatom].bonds[3] = 4;
                currentT += 0.5 * elements[allatoms[currentatom].type].mass
* (allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
                netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
                netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
                netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
                currentatom++;

                allatoms[currentatom].type = H;
                allatoms[currentatom].x = 1.5 * a + 3.2 * (double)i * a;
                allatoms[currentatom].y = 1.5 * a + 2.4 * (double)j * a;
                allatoms[currentatom].z = 1.5 * a + 3.8 * (double)k * a +
0.25 * a * (double)N;
                allatoms[currentatom].vx = ((double)rand()-RMO2)/RMO2;
                allatoms[currentatom].vy = ((double)rand()-RMO2)/RMO2;
                allatoms[currentatom].vz = ((double)rand()-RMO2)/RMO2;
                allatoms[currentatom].ax = 0.0;
                allatoms[currentatom].ay = 0.0;
                allatoms[currentatom].az = 0.0;
                allatoms[currentatom].ndx = 0.0;
                allatoms[currentatom].ndy = 0.0;
                allatoms[currentatom].ndz = 0.0;
                allatoms[currentatom].potential = 0.0;
                allatoms[currentatom].newpotential = 0.0;
                for(l=0; l<NUMBONDS; l++) allatoms[currentatom].bonds[l] =
0;
                allatoms[currentatom].bonds[0] = -1;
                currentT += 0.5 * elements[allatoms[currentatom].type].mass
* (allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);

```

```

allatoms[currentatom].vx;      netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;      netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;      netmz += elements[allatoms[currentatom].type].mass *
                                currentatom++;
                                allatoms[currentatom].type = H;
                                allatoms[currentatom].x = 1.5 * a + 3.2 * (double)i * a;
                                allatoms[currentatom].y = 2.5 * a + 2.4 * (double)j * a;
                                allatoms[currentatom].z = 2.5 * a + 3.8 * (double)k * a +
0.25 * a * (double)N;
                                allatoms[currentatom].vx = ((double)rand()-RMO2)/RMO2;
                                allatoms[currentatom].vy = ((double)rand()-RMO2)/RMO2;
                                allatoms[currentatom].vz = ((double)rand()-RMO2)/RMO2;
                                allatoms[currentatom].ax = 0.0;
                                allatoms[currentatom].ay = 0.0;
                                allatoms[currentatom].az = 0.0;
                                allatoms[currentatom].ndx = 0.0;
                                allatoms[currentatom].ndy = 0.0;
                                allatoms[currentatom].ndz = 0.0;
                                allatoms[currentatom].potential = 0.0;
                                allatoms[currentatom].newpotential = 0.0;
                                for(l=0; l<NUMBONDS; l++) allatoms[currentatom].bonds[l] =
0;
                                allatoms[currentatom].bonds[0] = -2;
                                currentT += 0.5 * elements[allatoms[currentatom].type].mass
* (allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
                                netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;      netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;      netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
                                currentatom++;
                                allatoms[currentatom].type = H;
                                allatoms[currentatom].x = 2.5 * a + 3.2 * (double)i * a;
                                allatoms[currentatom].y = 1.5 * a + 2.4 * (double)j * a;
                                allatoms[currentatom].z = 2.5 * a + 3.8 * (double)k * a +
0.25 * a * (double)N;
                                allatoms[currentatom].vx = ((double)rand()-RMO2)/RMO2;
                                allatoms[currentatom].vy = ((double)rand()-RMO2)/RMO2;
                                allatoms[currentatom].vz = ((double)rand()-RMO2)/RMO2;
                                allatoms[currentatom].ax = 0.0;
                                allatoms[currentatom].ay = 0.0;
                                allatoms[currentatom].az = 0.0;
                                allatoms[currentatom].ndx = 0.0;
                                allatoms[currentatom].ndy = 0.0;
                                allatoms[currentatom].ndz = 0.0;
                                allatoms[currentatom].potential = 0.0;
                                allatoms[currentatom].newpotential = 0.0;
                                for(l=0; l<NUMBONDS; l++) allatoms[currentatom].bonds[l] =
0;
                                allatoms[currentatom].bonds[0] = -3;
                                currentT += 0.5 * elements[allatoms[currentatom].type].mass
* (allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
                                netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;      netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;      netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
                                currentatom++;
                                for(n=0; n<N-1; n++)
                                {
* a + a*n;
* a + a*n;
* a + 0.25 * a * (double)N + 0.00*a*n;
                                allatoms[currentatom].type = C;
                                allatoms[currentatom].x = 2.5 * a + 3.2 * (double)i
                                allatoms[currentatom].y = 2.5 * a + 2.4 * (double)j
                                allatoms[currentatom].z = 1.5 * a + 3.8 * (double)k
                                allatoms[currentatom].vx = ((double)rand()-
RMO2)/RMO2;

```

```

allatoms[currentatom].vy = ((double)rand()-
RMo2)/RMo2;
allatoms[currentatom].vz = ((double)rand()-
RMo2)/RMo2;
allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;
allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++)
allatoms[currentatom].bonds[l] = 0;
allatoms[currentatom].bonds[0] = -4;
allatoms[currentatom].bonds[1] = 1;
allatoms[currentatom].bonds[2] = 2;
allatoms[currentatom].bonds[3] = 3;
currentT += 0.5 *
elements[allatoms[currentatom].type].mass *
(allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;
allatoms[currentatom].type = H;
allatoms[currentatom].x = 2.0 * a + 3.2 * (double)i
* a + a*n;
allatoms[currentatom].y = 3.0 * a + 2.4 * (double)j
* a + a*n;
allatoms[currentatom].z = 1.0 * a + 3.8 * (double)k
* a + 0.25 * a * (double)N + 0.00*a*n;
allatoms[currentatom].vx = ((double)rand()-
RMo2)/RMo2;
allatoms[currentatom].vy = ((double)rand()-
RMo2)/RMo2;
allatoms[currentatom].vz = ((double)rand()-
RMo2)/RMo2;
allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;
allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++)
allatoms[currentatom].bonds[l] = 0;
allatoms[currentatom].bonds[0] = -1;
currentT += 0.5 *
elements[allatoms[currentatom].type].mass *
(allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;
allatoms[currentatom].type = H;
allatoms[currentatom].x = 3.0 * a + 3.2 * (double)i
* a + a*n;
allatoms[currentatom].y = 2.0 * a + 2.4 * (double)j
* a + a*n;
allatoms[currentatom].z = 1.0 * a + 3.8 * (double)k
* a + 0.25 * a * (double)N + 0.00*a*n;
allatoms[currentatom].vx = ((double)rand()-
RMo2)/RMo2;
allatoms[currentatom].vy = ((double)rand()-
RMo2)/RMo2;
allatoms[currentatom].vz = ((double)rand()-
RMo2)/RMo2;

```

```

allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;
allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++)
allatoms[currentatom].bonds[l] = 0;
allatoms[currentatom].bonds[0] = -2;
currentT += 0.5 *
elements[allatoms[currentatom].type].mass *
(allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;
allatoms[currentatom].type = C;
allatoms[currentatom].x = 2.0 * a + 3.2 * (double)i
allatoms[currentatom].y = 2.0 * a + 2.4 * (double)j
allatoms[currentatom].z = 2.0 * a + 3.8 * (double)k
allatoms[currentatom].vx = ((double)rand()-
RmO2)/RmO2;
allatoms[currentatom].vy = ((double)rand()-
RmO2)/RmO2;
allatoms[currentatom].vz = ((double)rand()-
RmO2)/RmO2;
allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;
allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++)
allatoms[currentatom].bonds[l] = 0;
allatoms[currentatom].bonds[0] = -3;
allatoms[currentatom].bonds[1] = 1;
allatoms[currentatom].bonds[2] = 2;
allatoms[currentatom].bonds[3] = 3;
currentT += 0.5 *
elements[allatoms[currentatom].type].mass *
(allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;
allatoms[currentatom].type = H;
allatoms[currentatom].x = 1.5 * a + 3.2 * (double)i
allatoms[currentatom].y = 2.5 * a + 2.4 * (double)j
allatoms[currentatom].z = 2.5 * a + 3.8 * (double)k
allatoms[currentatom].vx = ((double)rand()-
RmO2)/RmO2;
allatoms[currentatom].vy = ((double)rand()-
RmO2)/RmO2;
allatoms[currentatom].vz = ((double)rand()-
RmO2)/RmO2;
allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;

```

```

allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++)
allatoms[currentatom].bonds[l] = 0;
allatoms[currentatom].bonds[0] = -1;
currentT += 0.5 *
elements[allatoms[currentatom].type].mass *
(allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;
allatoms[currentatom].type = H;
allatoms[currentatom].x = 2.5 * a + 3.2 * (double)i
allatoms[currentatom].y = 1.5 * a + 2.4 * (double)j
allatoms[currentatom].z = 2.5 * a + 3.8 * (double)k
allatoms[currentatom].vx = ((double)rand()-
allatoms[currentatom].vy = ((double)rand()-
allatoms[currentatom].vz = ((double)rand()-
allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;
allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++)
allatoms[currentatom].bonds[l] = 0;
allatoms[currentatom].bonds[0] = -2;
currentT += 0.5 *
elements[allatoms[currentatom].type].mass *
(allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;
}
allatoms[currentatom].type = C;
allatoms[currentatom].x = 2.5 * a + 3.2 * (double)i * a +
a*(N-1);
allatoms[currentatom].y = 2.5 * a + 2.4 * (double)j * a +
a*(N-1);
allatoms[currentatom].z = 1.5 * a + 3.8 * (double)k * a +
0.25 * a * (double)N + 0.00*a*(N-1);
allatoms[currentatom].vx = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].vy = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].vz = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;
allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++) allatoms[currentatom].bonds[l] =
0;
allatoms[currentatom].bonds[0] = -4;
allatoms[currentatom].bonds[1] = 1;
allatoms[currentatom].bonds[2] = 2;

```

```

allatoms[currentatom].bonds[3] = 3;
currentT += 0.5 * elements[allatoms[currentatom].type].mass
* (allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;

allatoms[currentatom].type = H;
allatoms[currentatom].x = 2.0 * a + 3.2 * (double)i * a +
a*(N-1);
allatoms[currentatom].y = 3.0 * a + 2.4 * (double)j * a +
a*(N-1);
allatoms[currentatom].z = 1.0 * a + 3.8 * (double)k * a +
0.25 * a * (double)N + 0.00*a*(N-1);
allatoms[currentatom].vx = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].vy = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].vz = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;
allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++) allatoms[currentatom].bonds[l] =
0;
allatoms[currentatom].bonds[0] = -1;
currentT += 0.5 * elements[allatoms[currentatom].type].mass
* (allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;

allatoms[currentatom].type = H;
allatoms[currentatom].x = 3.0 * a + 3.2 * (double)i * a +
a*(N-1);
allatoms[currentatom].y = 2.0 * a + 2.4 * (double)j * a +
a*(N-1);
allatoms[currentatom].z = 1.0 * a + 3.8 * (double)k * a +
0.25 * a * (double)N + 0.00*a*(N-1);
allatoms[currentatom].vx = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].vy = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].vz = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;
allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++) allatoms[currentatom].bonds[l] =
0;
allatoms[currentatom].bonds[0] = -2;
currentT += 0.5 * elements[allatoms[currentatom].type].mass
* (allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;

allatoms[currentatom].type = H;

```

```

a*(N-1);          allatoms[currentatom].x = 3.0 * a + 3.2 * (double)i * a +
a*(N-1);          allatoms[currentatom].y = 3.0 * a + 2.4 * (double)j * a +
0.25 * a * (double)N + 0.00*a*(N-1); allatoms[currentatom].z = 2.0 * a + 3.8 * (double)k * a +
allatoms[currentatom].vx = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].vy = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].vz = ((double)rand()-RMO2)/RMO2;
allatoms[currentatom].ax = 0.0;
allatoms[currentatom].ay = 0.0;
allatoms[currentatom].az = 0.0;
allatoms[currentatom].ndx = 0.0;
allatoms[currentatom].ndy = 0.0;
allatoms[currentatom].ndz = 0.0;
allatoms[currentatom].potential = 0.0;
allatoms[currentatom].newpotential = 0.0;
for(l=0; l<NUMBONDS; l++) allatoms[currentatom].bonds[l] =
0;
allatoms[currentatom].bonds[0] = -3;
currentT += 0.5 * elements[allatoms[currentatom].type].mass
* (allatoms[currentatom].vx*allatoms[currentatom].vx +
allatoms[currentatom].vy*allatoms[currentatom].vy +
allatoms[currentatom].vz*allatoms[currentatom].vz);
netmx += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vx;
netmy += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vy;
netmz += elements[allatoms[currentatom].type].mass *
allatoms[currentatom].vz;
currentatom++;
    }
}

currentT /= (double)numatoms;
currentT /= 1.5 * kB;

netmx /= (double)numatoms;
netmy /= (double)numatoms;
netmz /= (double)numatoms;

vMul = sqrt(TEMP/currentT);
for(i=0; i<numatoms; i++)
{
    while(allatoms[i].x > tempheader.bigX) allatoms[i].x -= tempheader.bigX;
    while(allatoms[i].x < 0.0) allatoms[i].x += tempheader.bigX;
    while(allatoms[i].y > tempheader.bigY) allatoms[i].y -= tempheader.bigY;
    while(allatoms[i].y < 0.0) allatoms[i].y += tempheader.bigY;
    while(allatoms[i].z > tempheader.bigZ) allatoms[i].z -= tempheader.bigZ;
    while(allatoms[i].z < 0.0) allatoms[i].z += tempheader.bigZ;

    allatoms[i].vx = (allatoms[i].vx - netmx/elements[allatoms[i].type].mass)
* vMul;
    allatoms[i].vy = (allatoms[i].vy - netmy/elements[allatoms[i].type].mass)
* vMul;
    allatoms[i].vz = (allatoms[i].vz - netmz/elements[allatoms[i].type].mass)
* vMul;
}

fwrite(&tempheader, sizeof(Headerinfo), 1, file);
fwrite(allatoms, sizeof(Atom), numatoms, file);

free(allatoms);
fclose(file);

return 1;
}

```

interact

Description

The **interact** application is the meat of the simulation implementation and takes the file generated by **makeoutfile** and runs the simulation. The details are discussed briefly in the Implementation section.

Code

interact.c

```
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "atoms.h"
#include "header.h"
#include "types.h"

#define getclosest(d,bo2,b) ((d > bo2)?(d-b):((d < -bo2)?(d+b):d))
#define OUTPUTEVERY 1000
#define MAXNEIGHBORS 500

const double kB = 1.380658E-23;

Atom* allatoms;
int numatoms;
int** neighbors;

double bigX,bigY,bigZ;
double bXo2,bYo2,bZo2;

double mincutoff;
double maxdr;

double dt,hdt;

double desiredT;
double T;
double velMul;

int iterations;

void correctT()
{
    int i;
    T = 0;
    for(i=0; i<numatoms; i++)
    {
        T += 0.5 * elements[allatoms[i].type].mass *
(allatoms[i].vx*allatoms[i].vx + allatoms[i].vy*allatoms[i].vy +
allatoms[i].vz*allatoms[i].vz);
    }

    T /= (double)numatoms;
    T /= 1.5 * kB;
}

void makeneighbors()
{
    int i,j,k;
    double dx,dy,dz;
    Atom* thisatom;
    Atom* thatatom;

    maxdr = 0.0;
```



```

printf("calculating neighbors at i=%d\n\n", iterations);
for(i=0; i<numatoms; i++)
{
    thisatom = allatoms + i;
    k = 0;

    thisatom->ndx = thisatom->x;
    thisatom->ndy = thisatom->y;
    thisatom->ndz = thisatom->z;

    for(j=i+1; j<numatoms && k<MAXNEIGHBORS; j++)
    {
        thatatom = allatoms + j;

        dx = thisatom->x - thatatom->x;
        dy = thisatom->y - thatatom->y;
        dz = thisatom->z - thatatom->z;

        dx = getclosest(dx,bXo2,bigX);
        dy = getclosest(dy,bYo2,bigY);
        dz = getclosest(dz,bZo2,bigZ);

        if(dx<mincutoff&&dx>-mincutoff&&dy<mincutoff&&dy>-
mincutoff&&dz<mincutoff&&dz>-mincutoff)
        {
            neighbors[i][k++] = j;
        }
    }

    neighbors[i][k] = -1;
}
}

void interactions()
{
    int i,j,k,l,m,n;
    Atom* thisatom;
    Atom* atom2;
    Atom* atom3;
    Atom* thatatom;
    int atom2off;
    int atom3off;
    int atom4off;
    double epsilon,sigma,thismass,thatmass,k0,x0,z;
    double rmsq;
    double dx,dy,dz;
    double rsq,rmor2,rmor6,rmor12,r,rmx0;
    double pot,mag;
    double fx,fy,fz;
    int bonded,doffset;

    for(i=0; i<numatoms; i++)
    {
        thisatom = allatoms + i;
        thismass = elements[thisatom->type].mass;

        /*order 1 bonded interactions*/
        for(j=0; j<NUMBONDS; j++)
        {
            if(thisatom->bonds[j] <= 0)
                continue;

            thatatom = thisatom + thisatom->bonds[j];
            thatmass = elements[thatatom->type].mass;

            switch(thisatom->type)
            {
            case H:
                k0 = 483;
                x0 = 1.12E-10;
                break;
            default:
                switch(thatatom->type)
                {
                case H:
                    k0 = 483;
                    x0 = 1.12E-10;
                    break;
                default:
                    k0 = 450;
                }
            }
        }
    }
}

```

```

        }
        }
        break;
    }

    dx = thisatom->x - thatatom->x;
    dy = thisatom->y - thatatom->y;
    dz = thisatom->z - thatatom->z;

    dx = getclosest(dx,bXo2,bigX);
    dy = getclosest(dy,bYo2,bigY);
    dz = getclosest(dz,bZo2,bigZ);

    rsq = dx*dx + dy*dy + dz*dz;
    r = sqrt(rsq);
    rmx0 = r-x0;

    pot = 0.5 * k0 * rmx0 * rmx0;
    mag = -k0 * rmx0 / r;
    fx = dx*mag;
    fy = dy*mag;
    fz = dz*mag;

    thisatom->newpotential += pot;
    thisatom->ax += fx/thismass;
    thisatom->ay += fy/thismass;
    thisatom->az += fz/thismass;

    thatatom->newpotential += pot;
    thatatom->ax -= fx/thatmass;
    thatatom->ay -= fy/thatmass;
    thatatom->az -= fz/thatmass;
}

/*order 2 bonded interactions*/
for(j=0; j<NUMBONDS; j++)
{
    atom2off = thisatom->bonds[j];
    if(atom2off == 0)
        continue;

    atom2 = thisatom+atom2off;

    for(k=0; k<NUMBONDS; k++)
    {
        atom3off = atom2off+atom2->bonds[k];
        if(atom2->bonds[k] == 0 || atom3off <= 0)
            continue;

        thatatom = thisatom + atom3off;
        thatmass = elements[thatatom->type].mass;

        switch(thisatom->type)
        {
            case H:
                switch(thatatom->type)
                {
                    case H:
                        x0 = 1.83E-10;
                        break;
                    default:
                        x0 = 2.18E-10;
                        break;
                }
                break;
            default:
                switch(thatatom->type)
                {
                    case H:
                        x0 = 2.18E-10;
                        break;
                    default:
                        x0 = 2.50E-10;
                        break;
                }
                break;
        }
    }
    k0 = 10;

    dx = thisatom->x - thatatom->x;
    dy = thisatom->y - thatatom->y;

```

```

dz = thisatom->z - thatatom->z;

dx = getclosest(dx,bXo2,bigX);
dy = getclosest(dy,bYo2,bigY);
dz = getclosest(dz,bZo2,bigZ);

rsq = dx*dx + dy*dy + dz*dz;
r = sqrt(rsq);
rmx0 = r-x0;

pot = 0.5 * k0 * rmx0 * rmx0;
mag = -k0 * rmx0 / r;
fx = dx*mag;
fy = dy*mag;
fz = dz*mag;

thisatom->newpotential += pot;
thisatom->ax += fx/thismass;
thisatom->ay += fy/thismass;
thisatom->az += fz/thismass;

thatatom->newpotential += pot;
thatatom->ax -= fx/thatmass;
thatatom->ay -= fy/thatmass;
thatatom->az -= fz/thatmass;
}
}

/*order 3 bonded interactions*/
for(j=0; j<NUMBONDS; j++)
{
atom2off = thisatom->bonds[j];
if(atom2off == 0)
continue;

atom2 = thisatom + atom2off;

for(k=0; k<NUMBONDS; k++)
{
atom3off = atom2off + atom2->bonds[k];
if(atom2->bonds[k] == 0 || atom3off <= 0)
continue;

atom3 = thisatom + atom3off;

for(l=0; l<NUMBONDS; l++)
{
atom4off = atom3off + atom3->bonds[l];
if(atom3->bonds[k] == 0 || atom4off <= 0)
continue;

thatatom = thisatom + atom4off;
thatmass = elements[thatatom->type].mass;

switch(thisatom->type)
{
case H:
switch(thatatom->type)
{
case H:
x0 = 2.278E-10;
z = 2.392E-19;
break;
default:
x0 = 2.446E-10;
z = 2.234E-19;
break;
}
break;
default:
switch(thatatom->type)
{
case H:
x0 = 2.446E-10;
z = 2.234E-19;
break;
default:
x0 = 2.551E-10;
z = 2.660E-19;
break;
}
}
}
}
}
}
}

```

```

        break;
    }

    dx = thisatom->x - thatatom->x;
    dy = thisatom->y - thatatom->y;
    dz = thisatom->z - thatatom->z;

    dx = getclosest(dx,bXo2,bigX);
    dy = getclosest(dy,bYo2,bigY);
    dz = getclosest(dz,bZo2,bigZ);

    rsq = dx*dx + dy*dy + dz*dz;

    pot = z*x0*x0/rsq;
    mag = 2.0*pot/rsq;
    fx = dx*mag;
    fy = dy*mag;
    fz = dz*mag;

    thisatom->newpotential += pot;
    thisatom->ax += fx/thismass;
    thisatom->ay += fy/thismass;
    thisatom->az += fz/thismass;

    thatatom->newpotential += pot;
    thatatom->ax -= fx/thatmass;
    thatatom->ay -= fy/thatmass;
    thatatom->az -= fz/thatmass;
    }
}

/*order 0, non-bonded (Van Der Waal's) interactions*/
for(j=0; j<MAXNEIGHBORS; j++)
{
    if(neighbors[i][j] == -1)
        break;

    thatatom = allatoms + neighbors[i][j];
    doffset = neighbors[i][j] - i;

    bonded = 0;
    for(n=0; n<NUMBONDS; n++)
    {
        if(doffset == thisatom->bonds[n])
            bonded = 1;

        for(m=0; m<NUMBONDS; m++)
        {
            if(doffset == (thisatom+thisatom->bonds[n])-
>bonds[m]+thisatom->bonds[n])
                bonded = 1;
        }
    }

    if(bonded)
        continue;

    thatmass = elements[thatatom->type].mass;
    fx = fy = fz = 0.0;

    if(thisatom->type == thatatom->type)
    {
        epsilon = elements[thisatom->type].epsilon;
        sigma = 2.0 * elements[thisatom->type].radius;
    }
    else
    {
        epsilon = sqrt(elements[thisatom->
>type].epsilon*elements[thatatom->type].epsilon);
        sigma = elements[thisatom->type].radius +
elements[thatatom->type].radius;
    }

    rmsq = sigma*sigma;

    dx = thisatom->x - thatatom->x;
    dy = thisatom->y - thatatom->y;
    dz = thisatom->z - thatatom->z;
}

```

```

        dx = getclosest(dx,bXo2,bigX);
        dy = getclosest(dy,bYo2,bigY);
        dz = getclosest(dz,bZo2,bigZ);

        rsq = dx*dx + dy*dy + dz*dz;

        rmor2 = rmsq/rsq;
        rmor6 = rmor2*rmor2*rmor2;
        rmor12 = rmor6*rmor6;

        pot = -4.0*epsilon*(rmor6-rmor12);
        mag = -24.0*(epsilon/rsq)*(rmor6-2.0*rmor12);
        fx = dx*mag;
        fy = dy*mag;
        fz = dz*mag;

        thisatom->newpotential += pot;
        thisatom->ax += fx/thismass;
        thisatom->ay += fy/thismass;
        thisatom->az += fz/thismass;

        thatatom->newpotential += pot;
        thatatom->ax -= fx/thatmass;
        thatatom->ay -= fy/thatmass;
        thatatom->az -= fz/thatmass;
    }
}

void integrate()
{
    int i;
    double axhdt, ayhdt, azhdt;
    double dx,dy,dz;
    Atom* thisatom;

    for(i=0; i<numatoms; i++)
    {
        thisatom = allatoms + i;

        thisatom->potential = thisatom->newpotential;

        /*temperature correction*/
        thisatom->vx *= velMul;
        thisatom->vy *= velMul;
        thisatom->vz *= velMul;

        /* velocity verlet integration */
        axhdt = thisatom->ax * hdt;
        ayhdt = thisatom->ay * hdt;
        azhdt = thisatom->az * hdt;
        thisatom->vx += axhdt;
        thisatom->vy += ayhdt;
        thisatom->vz += azhdt;
        thisatom->x += thisatom->vx * dt + axhdt*dt;
        thisatom->y += thisatom->vy * dt + ayhdt*dt;
        thisatom->z += thisatom->vz * dt + azhdt*dt;
        thisatom->vx += axhdt;
        thisatom->vy += ayhdt;
        thisatom->vz += azhdt;

        if(thisatom->x > bigX) thisatom->x -= bigX;
        else if(thisatom->x < 0.0) thisatom->x += bigX;
        if(thisatom->y > bigY) thisatom->y -= bigY;
        else if(thisatom->y < 0.0) thisatom->y += bigY;
        if(thisatom->z > bigZ) thisatom->z -= bigZ;
        else if(thisatom->z < 0.0) thisatom->z += bigZ;

        dx = thisatom->x - (thisatom->ndx);
        dy = thisatom->y - (thisatom->ndy);
        dz = thisatom->z - (thisatom->ndz);

        dx = getclosest(dx,bXo2,bigX);
        dy = getclosest(dy,bYo2,bigY);
        dz = getclosest(dz,bZo2,bigZ);

        maxdr = max(max(maxdr,dx),max(dy,dz));;

        thisatom->newpotential = 0.0;
        thisatom->ax = 0.0;
        thisatom->ay = 0.0;
    }
}

```

```

        thisatom->az = 0.0;
    }
}

int main(int argc, char** argv)
{
    int i,j;
    int outevery = OUTPUTEVERY;
    int keepgoing;
    Headerinfo tempheader;
    FILE* file;
    int outnum;
    char buffer[15];
    int time0;

    time0 = (int)time(NULL);

    outnum = 0;

    if(argc < 2)
    {
        printf("no infile specified\n\n");
        return 0;
    }

    file = fopen(argv[1], "rb");

    if(file == NULL)
    {
        printf("failed to open infile %s for reading\n\n", argv[1]);
        return errno;
    }

    printf("using infile %s\n\n", argv[1]);

    i = (int)fread(&tempheader, sizeof(Headerinfo), 1, file);

    if(i < 1)
    {
        printf("error reading header from infile\n\n");
        return 0;
    }

    iterations = tempheader.iterations;

    if(iterations > 0)
        keepgoing = 1;
    else
        keepgoing = 0;

    mincutoff = tempheader.mincutoff;
    dt = tempheader.dt;
    bigX = tempheader.bigX;
    bXo2 = bigX/2.0;
    bigY = tempheader.bigY;
    bYo2 = bigY/2.0;
    bigZ = tempheader.bigZ;
    bZo2 = bigZ/2.0;
    desiredT = tempheader.desiredT;
    T = desiredT;

    numatoms = tempheader.numatoms;

    allatoms = (Atom*)malloc(numatoms*sizeof(Atom));
    if(allatoms == NULL)
    {
        printf("unable to allocate memory for allatoms\n\n");
        return 0;
    }

    i = (int)fread(allatoms, sizeof(Atom), numatoms, file);

    if(i < numatoms)
    {
        printf("error reading atoms from infile\n\n");
        return 0;
    }

    fclose(file);

    neighbors = (int**)malloc(numatoms*sizeof(int*));

```

```

if(neighbors == NULL)
{
    printf("unable to allocate memory for neighbors\n\n");
    free(allatoms);
    return 0;
}
for(i=0; i<numatoms; i++)
{
    neighbors[i] = (int*)malloc(MAXNEIGHBORS*sizeof(int));
    if(neighbors[i] == NULL)
    {
        printf("unable to allocate memory for neighbors[%d]\n\n", i);
        for(j=0; j<i; j++)
        {
            free(neighbors[i]);
        }
        free(neighbors);
        free(allatoms);
        return 0;
    }
}
for(i=0; i<numatoms; i++)
{
    for(j=0; j<MAXNEIGHBORS; j++)
    {
        neighbors[i][j] = -1;
    }
}
makeneighbors();
hdt = 0.5 * dt;
while(keepgoing)
{
    if(maxdr>0.5*mincutoff)
    {
        makeneighbors();
    }

    interactions();

    correctT();
    velMul = sqrt(desiredT/T);

    integrate();

    iterations--;

    if(T>10000)
    {
        iterations = min(10, iterations);
        outevery = 1;
    }

    if(iterations == outevery*(iterations/outevery))
    {
        file = NULL;
        do
        {
            if(file != NULL) fclose(file);
            outnum++;
            sprintf(buffer, "o%07d.atm", outnum);
            file = fopen(buffer, "r");
        } while(file != NULL);
        printf("%s\n", buffer);
        printf("T=%f\n", T);
        printf("i=%d\n", iterations);
        printf("t=%d\n", (int)time(NULL) - time0);
        printf("\n");
        file = fopen(buffer, "wb");

        tempheader.iterations = iterations;
        tempheader.mincutoff = mincutoff;
        tempheader.dt = dt;
        tempheader.bigX = bigX;
        tempheader.bigY = bigY;
        tempheader.bigZ = bigZ;
        tempheader.numatoms = numatoms;
        tempheader.desiredT = desiredT;
        tempheader.T = T;
    }
}

```

```

        fwrite(&tempheader, sizeof(Headerinfo), 1, file);
        fwrite(allatoms, sizeof(Atom), numatoms, file);
        fclose(file);
    }
    if(iterations > 0)
        keepgoing = 1;
    else
        keepgoing = 0;
}
free(allatoms);
return 1;
}

```

atm2hr

Description

The **atm2hr** application will take simulation data files and convert them to a human readable format. The output is incredibly verbose and tends to be difficult to wade through but is more informative than the binary data files.

Code

atm2hr.c

```

#include <errno.h>
#include <stdio.h>
#include "atoms.h"
#include "header.h"

int main(int argc, char** argv)
{
    Headerinfo tempheader;
    Atom tempatom;
    FILE* file1;
    FILE* file2;
    int i,j,k;
    char buffer[255];

    if(argc < 2)
    {
        printf("no infile specified\n\n");
        return 0;
    }

    for(k=1; k<argc; k++)
    {
        file1 = fopen(argv[k], "rb");

        if(file1 == NULL)
        {
            printf("failed to open infile %s for reading\n\n", argv[k]);
            return errno;
        }

        printf("using infile %s\n\n", argv[k]);
        j = (int)fread(&tempheader, sizeof(Headerinfo), 1, file1);
        if(j < 1)
        {
            printf("error reading header from infile\n\n");
            return 0;
        }

        sprintf(buffer, "%s.hr", argv[k]);
        file2 = fopen(buffer, "w");
    }
}

```



```

fprintf(file2, "iterations remaining = %d\n", tempheader.iterations);
fprintf(file2, "minimum cutoff distance = %e\n", tempheader.mincutoff);
fprintf(file2, "delta time = %e\n", tempheader.dt);
fprintf(file2, "system x size = %e\n", tempheader.bigX);
fprintf(file2, "system y size = %e\n", tempheader.bigY);
fprintf(file2, "system z size = %e\n", tempheader.bigZ);
fprintf(file2, "number of atoms = %d\n", tempheader.numatoms);
fprintf(file2, "\n");

for(i=0; i<tempheader.numatoms; i++)
{
    j = (int)fread(&tempatom, sizeof(Atom), 1, file1);
    if(j < 1)
    {
        printf("error reading atom %d from infile\n\n", i);
        return 0;
    }

    fprintf(file2, "Atom #%d\n", i);
    fprintf(file2, "type = %s\n", elements[tempatom.type].symbol);
    fprintf(file2, "position = <%e, %e, %e>\n", tempatom.x, tempatom.y,
tempatom.z);
    fprintf(file2, "velocity = <%e, %e, %e>\n", tempatom.vx,
tempatom.vy, tempatom.vz);
    fprintf(file2, "potential = %e\n", tempatom.potential);
    fprintf(file2, "\n");
}

fclose(file2);
fclose(file1);
}

return 1;
}

```

atm2pov

Description

The **atm2pov** application will take simulation data files and convert them to a format which can be used by the freely available application **POV-Ray** (<http://www.povray.org/>) to generate still images from any frame of the simulation. Although purely qualitative in nature, the images are very useful for getting a good idea of what is going on.

Code

atm2pov.c

```

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include "atoms.h"
#include "header.h"

int main(int argc, char** argv)
{
    Headerinfo tempheader;
    Atom tempatom;
    FILE* file1;
    FILE* file2;
    int i,j,k;
    char buffer[255];
    double sizemul;
    double cutoff;
    double curX, curY, curZ;

    if(argc < 2)
    {
        printf("no infile specified\n\n");
        return 0;
    }
}

```

```

for(k=1; k<argc; k++)
{
    file1 = fopen(argv[k], "rb");
    if(file1 == NULL)
    {
        printf("failed to open infile %s for reading\n\n", argv[k]);
        return errno;
    }
    printf("using infile %s\n\n", argv[k]);
    j = (int)fread(&tempheader, sizeof(Headerinfo), 1, file1);
    if(j < 1)
    {
        printf("error reading header from infile\n\n");
        return 0;
    }
    sprintf(buffer, "%s.pov", argv[k]);
    file2 = fopen(buffer, "w");
    sizemul = max(tempheader.bigX, max(tempheader.bigY, tempheader.bigZ));
    if(sizemul <= 0.0)
    {
        fprintf(file2, "crash HARD and die\n");
        fclose(file2);
        fclose(file1);
        return 5;
    }
    sizemul = 1/sizemul;
    cutoff = tempheader.mincutoff * sizemul;
    curX = tempheader.bigX*sizemul;
    curY = tempheader.bigY*sizemul;
    curZ = tempheader.bigZ*sizemul;
    fprintf(file2, "#include \"colors.inc\"\n\n");
    fprintf(file2, "camera {\n\tlocation <f, %f, %f>\n\tlook_at <f, %f, %f>\n\n", 0.1+0.5*curX, 0.7+0.5*curY, -(0.8+0.5*curZ), 0.5*curX, 0.5*curY, 0.5*curZ);
    fprintf(file2, "light_source { <10, 10, -10>, White }\n\n");
    fprintf(file2, "background { color Black }\n\n");
    fprintf(file2, "cylinder { <0.0, -0.025, -0.025>, <0.0, -0.025, -0.025> + <f, 0.0, 0.0>, 0.01 pigment { Green } }\n\n", 1.0E-9*sizemul);
    fprintf(file2, "cylinder { <0.0, -0.025, -0.025>, <0.0, -0.025, -0.025> + <f, 0.0, 0.0>, 0.011 pigment { GreenCopper } }\n\n", 1.0E-10*sizemul);
    fprintf(file2, "cylinder { <0.0, 0.0, 0.0>, <f, 0.0, 0.0>, 0.002 pigment { Red } }\n", curX);
    fprintf(file2, "cylinder { <0.0, 0.0, 0.0>, <0.0, %f, 0.0>, 0.002 pigment { Red } }\n", curY);
    fprintf(file2, "cylinder { <0.0, 0.0, 0.0>, <0.0, 0.0, %f>, 0.002 pigment { Red } }\n", curZ);
    fprintf(file2, "cylinder { <f, 0.0, 0.0>, <f, 0.0, %f>, 0.002 pigment { Red } }\n", curX, curX, curZ);
    fprintf(file2, "cylinder { <f, 0.0, 0.0>, <f, %f, 0.0>, 0.002 pigment { Red } }\n", curX, curX, curY);
    fprintf(file2, "cylinder { <0.0, %f, 0.0>, <0.0, %f, %f>, 0.002 pigment { Red } }\n", curY, curY, curZ);
    fprintf(file2, "cylinder { <0.0, %f, 0.0>, <f, %f, 0.0>, 0.002 pigment { Red } }\n", curY, curX, curY);
    fprintf(file2, "cylinder { <0.0, 0.0, %f>, <f, 0.0, %f>, 0.002 pigment { Red } }\n", curZ, curX, curZ);
    fprintf(file2, "cylinder { <0.0, 0.0, %f>, <0.0, %f, %f>, 0.002 pigment { Red } }\n", curZ, curY, curZ);
    fprintf(file2, "cylinder { <f, %f, %f>, <0.0, %f, %f>, 0.002 pigment { Red } }\n", curX, curY, curZ, curY, curZ);
    fprintf(file2, "cylinder { <f, %f, %f>, <f, 0.0, %f>, 0.002 pigment { Red } }\n", curX, curY, curZ, curX, curZ);
    fprintf(file2, "cylinder { <f, %f, %f>, <f, %f, 0.0>, 0.002 pigment { Red } }\n", curX, curY, curZ, curX, curY);
    fprintf(file2, "\n\n");
}

```

```

        for(i=0; i < (sizeof(elements)/sizeof(elements[0])); i++)
        {
            fprintf(file2, "#macro %s(center)\n", elements[i].symbol);
            fprintf(file2, "\tsphere { center, %f pigment { %s } }\n",
elements[i].radius*sizemul, elements[i].pigment);
            fprintf(file2, "#end\n");
        }
        fprintf(file2, "\n");
        for(i=0; i<tempheader.numatoms; i++)
        {
            j = (int)fread(&tempatom, sizeof(Atom), 1, file1);
            if(j < 1)
            {
                printf("error reading atom %d from infile\n\n", i);
                return 3;
            }

            curX = sizemul * tempatom.x;
            curY = sizemul * tempatom.y;
            curZ = sizemul * tempatom.z;

            fprintf(file2, "%s( <%f,%f,%f> )\n",
elements[tempatom.type].symbol, curX, curY, curZ);
        }

        fclose(file2);
        fclose(file1);
    }

    return 1;
}

```