

Computational and Graphical Modeling of Physical Systems

Nicholas Krainak

May 13, 2004

1 1.0 Introduction

The purpose of this project is to develop software that can efficiently model physical simulations using input from the user, such as a variety of differential equations to model one or many particles interacting. Research areas include methods of error reduction, as well as many different physical areas, ranging from classical mechanics to fluid dynamics. This project will require me to research many physical models and computational methods that produce a result within a reasonable error. It will also require me to ensure that these answers are available within a reasonable amount of time after starting the simulation. Parties interested modeling a system could use the software to produce results in a desired format, and the software could also graphically model the simulations. This project could also be used as a teaching aid for physics students.

2 2.0 Background

Theoretical physicists often develop differential equations to which there are no known solutions. As such, there is no absolute quantitative way of knowing exactly how a particle will act under such circumstances, though it is necessary to know how a particle will behave. Physical computation of simulations are one way to model a particle. A computational physicist must be willing to say with complete confidence that their model is loyal to the equations that are being modeled. Additionally, computational modeling is advantageous for use in modeling large systems that would be cumbersome or impossible to do by hand; for instance, a large number of molecules. For these reasons, it is imperative to have a working program that will ensure that the model is accurate to within a specific degree of error.

3 3.0 Theory

Overview: This project will be incorporating several algorithms, and will be able to use them for various situations. The first algorithm that I am hoping to implement will most

likely be the most straightforward and computationally efficient, but will also have the most room for error. This algorithm is called the *leapfrog* algorithm, which is used for differential equations of the second order.

3.1 Leapfrog

The first method used is the *leapfrog* algorithm, which is a modified version of the *Verlet* algorithm. The *Verlet* algorithm uses the positions and accelerations at the time t and the positions at the time

$$t - \Delta t. \tag{1}$$

to predict the positions at the time $t + \Delta t$ where is the integration step. From a Taylor expansion of the 3-rd order, we obtain

$$\mathbf{r}_i(t + \Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t - \Delta t) + \ddot{\mathbf{r}}_i(t)\Delta t^2. \tag{2}$$

The error in the atomic positions is of the order of Δt^4 . The velocities are obtained from the basic definition of differentiation

$$\dot{\mathbf{r}}_i(t) = \frac{\mathbf{r}_i(t + \Delta t) - \mathbf{r}_i(t - \Delta t)}{2\Delta t}, \tag{3}$$

with an error of the order of Δt^2 . To obtain more accurate velocities, the *leapfrog* algorithm is used, using velocities at half time step

$$\dot{\mathbf{r}}_i(t + \frac{\Delta t}{2}) = \dot{\mathbf{r}}_i(t - \frac{\Delta t}{2}) + \ddot{\mathbf{r}}_i(t)\Delta t. \tag{4}$$

The velocities at time t can be also computed from

$$\dot{\mathbf{r}}_i(t) = \frac{\dot{\mathbf{r}}_i(t + \frac{\Delta t}{2}) + \dot{\mathbf{r}}_i(t - \frac{\Delta t}{2})}{2}. \tag{5}$$

This is useful when the kinetic energy is needed at time t , as for example in the case where velocity rescaling must be carried out (see below). The positions are then obtained from

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \dot{\mathbf{r}}_i(t + \frac{\Delta t}{2})\Delta t. \tag{6}$$

The *leapfrog* algorithm is computationally less expensive than the Predictor-Corrector approach for example, and requires less storage. This could be an important advantage in the case of large scale calculations. Moreover, the conservation of energy is respected, even at large time steps. Therefore, the computation time could be greatly decreased when this algorithm is used. However, when more accurate velocities and positions are needed, another algorithm should be implemented, like the Predictor-Corrector algorithm. (Will be detailed at a later date).

3.2 Cell Model Computation

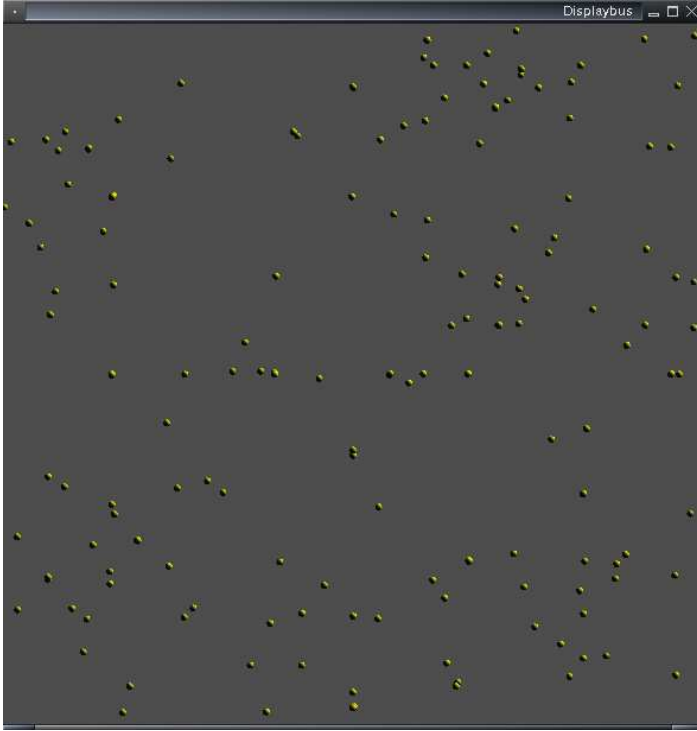
Most gaseous systems will have many millions of particles interacting. Given today's technology, calculating the forces on each particle due to every other is highly impracticable. Luckily, the forces between particles are only significantly strong when the particles are close to each other. Taking this into account, we can greatly speed up the process of calculation by dividing up the volume or area into many different cells, such as in the shape of a cube, each containing only a few dozen or so particles. Then calculations for the next position of each particle are performed while only calculating the forces on the particle due only to the particles in the same and adjacent cubes.

For example, a system of many millions of particles exists, but you are only interested in the gross behavior of it as a whole, such as average kinetic energy (temperature) or pressure, or some other such physical quantity. Using the cell model computation method, you divide the system up into small enough sections so that each holds particles numbering on the order of a few hundred. Then, take one average cell, and proceed as normal, copying that cell 26 times, making it the center of a 3x3 cube of such cells. And each particle in the center cell interacts with all of the other particles of all 27 cells, which creates, with random data, a good estimate of how the system acts as a whole. When a particle leaves the boundary of the cell, it wraps to the opposite side, simulating particles leaving and entering the system.

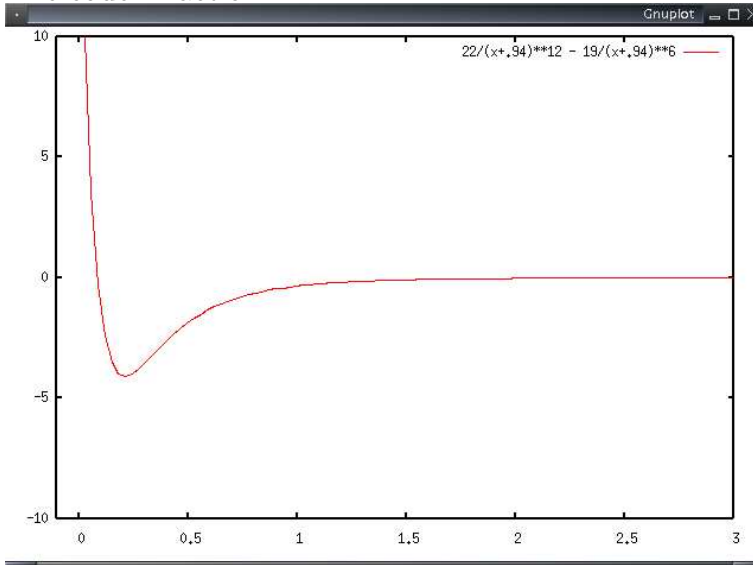
3.3 Dipole Interactions Among Gaseous Particles

When investigating the motion of molecules in a gaseous system, one needs an appropriate force and interaction potential energy that is very small when far away, mildly attractive when an appropriate distance away, and very repulsive when in very close proximity to other molecules. A scientist has come up with a potential energy function, called the six-twelve potential, that roughly models the interactions among molecules using a mock-up of a dipole force that takes the form of $\frac{a}{r^{12}} - \frac{b}{r^6}$ where r is the distance between the two particles and a and b are constants.

3.4 Pictures



The code in action.



A typical six-twelve potential.

3.5 Extentions The leapfrog algorithm can be used in many other areas than N-body systems. It can be used in continuum systems as well. Given a region with specific boundary conditions, specifically the boundary is fixed, the second derivative of the boundary position is zero, or there is no boundary (i.e. a topological sphere) , the region within the boundaries will converge to LaPlace's wave equation using a relatively simple algorithm. This algorithm involves averaging the position in the $N+1$ dimension (with the region being in N dimensions) with that of it's neighbors. It turns out that, using a variation of this algorithm (which will

be derived at a later date) combined with the Leapfrog algorithm, it is possible to model waves that approximate LaPlace's equation. This method, which I am currently developing, will be flushed out in greater detail at a future date when I have more time.

3.6 Code

The Leapfrog Algorithm at work in the Cell Model:

```
void leapfrog()
{
for(int i=0; i<N; i++) //for each particle
{
for(int k=0; k<N; k++) //for each other particle
{
for(int z=-1; z<=1; z++) // for each horizontal layer
{
for(int y=-1; y<=1; y++) // for each sidetoside layer
{
for(int x=-1; x<=1; x++) //for each fronttoback layer
{
vector fArray = calcacc(data[curstep-1][i].rx, data[curstep-1][i].ry, data[curstep-1][i].rz);
data[curstep-1][i].ax+=fArray.array[0]/massArray[i]; //update acceleration
data[curstep-1][i].ay+=fArray.array[1]/massArray[i];
data[curstep-1][i].az+=fArray.array[2]/massArray[i];
if(!x && !y && !z) //if k particle is in middle cell
{
data[curstep-1][k].ax+=-1.0*fArray.array[0]/massArray[k]; //use Newton's 2nd Law
data[curstep-1][k].ay+=-1.0*fArray.array[1]/massArray[k];
data[curstep-1][k].az+=-1.0*fArray.array[2]/massArray[k];
}
}
}
}
}
}
for(int j=0; j<N; j++)
{
data[curstep][j].vx=data[curstep-1][j].vx+data[curstep-1][j].ax*timestep; //update velocity
data[curstep][j].vy=data[curstep-1][j].vy+data[curstep-1][j].ay*timestep;
data[curstep][j].vz=data[curstep-1][j].vz+data[curstep-1][j].az*timestep;

data[curstep][j].rx=data[curstep-1][j].rx+data[curstep][j].vx*timestep+0.5*data[curstep-1][j].ax*timestep*timestep;
data[curstep][j].ry=data[curstep-1][j].ry+data[curstep][j].vy*timestep+0.5*data[curstep-1][j].ay*timestep*timestep;
data[curstep][j].rz=data[curstep-1][j].rz+data[curstep][j].vz*timestep+0.5*data[curstep-1][j].az*timestep*timestep;
}
```

```
//Now Check Boundaries, wrap if necessary.
```

```
if(data[curstep][j].rx<-1.0*CUBESIDE)
{
collisions++;
data[curstep][j].rx*=-1.0;
while(data[curstep][j].rx>CUBESIDE)
data[curstep][j].rx-=0.1;
}
else if(data[curstep][j].rx>CUBESIDE)
{
collisions++;
data[curstep][j].rx*=-1.0;
while(data[curstep][j].rx<-1.0*CUBESIDE)
data[curstep][j].rx+=0.1;
}
```

```
if(data[curstep][j].ry<-1.0*CUBESIDE)
{
collisions++;
data[curstep][j].ry*=-1.0;
while(data[curstep][j].ry>CUBESIDE)
data[curstep][j].ry-=0.1;
}
else if(data[curstep][j].ry>CUBESIDE)
{
collisions++;
data[curstep][j].ry*=-1.0;
while(data[curstep][j].ry<-1.0*CUBESIDE)
data[curstep][j].ry+=0.1;
}
```

```
if(data[curstep][j].rz<-1.0*CUBESIDE)
{
collisions++;
data[curstep][j].rz*=-1.0;
while(data[curstep][j].rz>CUBESIDE)
data[curstep][j].rz-=0.1;
}
else if(data[curstep][j].rz>CUBESIDE)
```

```

{
collisions++;
data[curstep][j].rz*=-1.0;
while(data[curstep][j].rz<-1.0*CUBESIDE)
data[curstep][j].rz+=0.1;
}
}

```

Here is a sample of the Wave equation algorithm using the variant with the Leapfrog Algorithm:

```

void average(int i, int k)
{
acc[i][k]= -1.0*wave[i][k] + c*(wave[i-1][k]+wave[i+1][k]+wave[i][k-1]+wave[i][k+1])/4;
vel[i][k]=vel[i][k] + acc[i][k]*dt; //update velocity for region in the continuum
wave[i][k]=wave[i][k]+vel[i][k]*dt + 0.5*acc[i][k]*dt*dt; //update position for region i
}

```

4 4.0 Quantum Mechanical Applications and Computational Models

4.1 What is Quantum Mechanics? Quantum mechanics differs from classical mechanics in that while in classical mechanics deals with definite places and various derivatives thereof, quantum mechanics deals with probability distributions throughout a given space.

Fortunately, quantum mechanics is very easily (if complexly) modeled on a computer. Probability distributions exist most often in quantized or discrete space and thusly there is no error accrued while attempting to model things on a continuum as there is in classical mechanics or N-body systems. One models motion of wave packets through space. A wave packet can be simply described as the probability distribution that describes the location of a particle in space. Thusly, one can quantize space (easily done with an array or matrix space) and evolve a particle's motion through space by accurately modifying the probability distribution. A check for the accuracy of this method can be performed by using the odd principle that in quantum mechanics, the sum of each individual probability squared will add up to one. That is, $P(0)^2 + P(1)^2 + \dots + P(N)^2 = 1$.

Given the state of a system at t and the forces and constraints to which it is subject, there is an equation, "Schrödinger's equation", that gives the state at any other time $U - vt_i - j - vt'_i$. The important properties of U for our purposes are that it is deterministic, which is to say that it takes the state of a system at one time into a unique state at any other, and it is linear, which is to say that if it takes a state $-A_i$ onto the state $-A_i$, and it takes the state $-B_i$ onto the state $-B_i$, then it takes any state of the form $-A_i + -B_i$ onto the state $-A_i + -B_i$.

Explicitly, the Shrodinger equation that satisvies the wave function $\psi(\mathbf{r}, t)$ of a particle moving in one dimation is given by

$$i\hbar \frac{\partial \psi}{\partial t} = \frac{-\hbar^2}{2m} \left(\frac{\partial^2}{\partial x^2} \right) \psi \quad (7)$$

This reduces to

$$\frac{\partial \psi}{\partial t} = \frac{-\hbar}{i * 2m} \left(\frac{\partial^2}{\partial x^2} \right) \psi \quad (8)$$

4.2 Modelling a Wave Packet using a Leapfrog Algorithm Variant

To advance the Schrdinger equation through time in discrete space, one first dimensionally reduces the equation, which is an elementary exercise. Next, one places the initial ψ values into a column vector, and generates an evolution matrix starting with the simple tri-diagonal matrix derived from an analysis of classical mechanics; that is, a matrix where the diagonal consists of negative two's, the elements next to the diagonals are one's, and all the other elements are zero. Using taylor series, this matrix, let's call it A is evolved in the form

$$\psi(t + \delta t) = \psi(t) * (1 + \frac{\psi(\Delta t)^1}{1!} + \frac{\psi(\Delta t)^2}{2!} + \frac{\psi(\Delta t)^3}{3!} + \dots) \quad (9)$$

where you substitute the matrix A and evolve for as long as you want; the more iterations of this algorithm that you implement, the more accurate it will be. After this matrix is generated, you run the evolution much like the leapfrog algorithm; that is

$$\psi(t + \Delta t) = e^{(i * \theta)} * A * \psi(t) \quad (10)$$

4.3 Modelling a Wave Packet using in a numerically accurate fashion

As it turns out, the Schrdinger can be made to evolve to the numerical accuracy of the computer using basic linear algebra techniques. If you take the matrix A described above, and instead of doing a taylor series expansion on it, you find the eigenvalues of the matrix, and, taking those finding the basis of the eigenspace, you have a method for time-step independent accuracy. The evolution through time can then be described thusly:

$$\psi(t) = \sum constant * e^{(i * eigenvalue)} * Eigenvector * psi(initial) \quad (11)$$

where the constants are determined by solving a linear system on the inital ψ values

5 5.0 Design

To develop this software, I will go on a stepwise basis. I will start out coding in C++, because that is my most comfortable language, and later translate the code to PERL so that a more flexible program will be possible, with users inputing force functions and potential energy functions that are readable by PERL. I will start out using just the *leapfrog* algorithm to get a basic simulation going, and I will then help to constrain the simulation from going

outside of acceptable ranges for calculations by taking into account the conservation laws; namely energy, linear momentum, and angular momentum. I will also simultaneously be working on developing code to display the results of the simulation in OpenGL. The display code will be able to display things in either two or three dimensions. Furthermore, all of this will be eventually controlled by the user, through means of a textfile, until such time should occur that I decide a graphical interface is necessary.

As to ensuring a good computation, the conservation laws (conservation of energy, linear momentum, and angular momentum) will all be taken into account. It will be arranged such that the user will input an acceptable deviation percentage from the initial conditions and as the computation progresses, these attributes will not be allowed to deviate from these limits, through such features as a variable timestep.

6 5.0 References

Oberacker, Volker E. "Computational Physics." 24 July 2001. Vanderbilt University. 19 September 2003 <
http://www.physics.vanderbilt.edu/volker/research/computat_physics/eo.html > .

Saada, David. "Equations of Motion." Israel Institute of Technology. 19 September 2003 [<http://phycomp.technion.ac.il/~david/thesis/node34.html>].

<http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/schr.html>

<http://web.hamline.edu/depts/chemistry/ccreswel/education/qm/Schrodinger.html>

Vesely, Franz J. "Computational Physics Algorithms." Differential Equations of the Second Order. University of Vienna. 19 September 2003 [<http://www.ap.univie.ac.at/users/ves/cp0102/dx/no>].