Ideal Gas Simulation

with n-body collisions

Alden Chad Daniels
Marlboro College

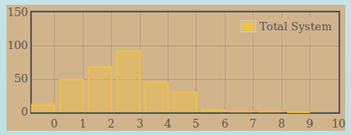
Data

Velocity Distribution Data:

In order to gather information on the velocities of the particles during the simulation a function is called during the loop through the particles which increments values in a temporary array according to how many particles fall within each range of velocity. After every full loop through the particles this temporary array is copied to the actual velocity range array which is what is read into the distribution graph as data.

```
function sortVelocity(velocity){
    var vRange = parseFloat($("#ball_speed").html());
    var tick_range = vRange/tick_count;
    for(j=0;j<=tick_count;j++){
        var velMag = vec3.length(velocity);
        if( (velMag >= (2*(j) * tick_range))&&
            (velMag < (2*(j+1) * tick_range))){
                temp_array[j] += 1;
        }
    }
}
```

This means that the velocity distribution information is effectively updated once for every loop through the particles. Since the graphical box display itself is only updated once for every loop the information keeps pace and appears relatively seamlessly



Here the y axis represents number of particles, while the x axis represents the magnitude of particle's velocity

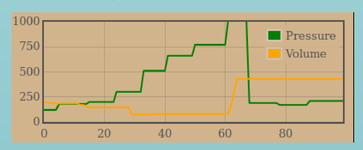
Pressure-Volume Data:

The volume is controlled by the user so its value is gathered trivially and the data is updated instantly whenever it is changed by the user.

Pressure on the other hand is more difficult. The pressure data is gathered by measuring the changes in momentum that occur whenever a particle collides with the walls of the container. To keep the updates accurate the data is gathered over a specified time interval which is 2 seconds by default. The changes are added to a temporary value which is copied to the total every interval. Finally the pressure is updated by dividing this total by the interval and the surface area of the box.

```
function updateStats(){
    var currTime = new Date().getTime();
    var elapsedTime = currTime - sinceInterval;
    var runTime = currTime - startTime;
    var interval = 2; // in seconds
    $("#run_time").html(Math.round(runTime/1000));
    if(elapsedTime >= interval * 1000){
        var tot_impulse = getRecentImpulse();
        var side_length = parseFloat($("#box_length").html());
        var compression = parseFloat($("#compression").html());
        var surface_area = 2*side_length*side_length+(4*side_length*(side_length+compression));
        var presConvFactor = .001;
        $("#box_area").html(Math.round(surface_area));
        $("#box_pressure").html(Math.round(tot_impulse/(interval*surface_area*presConvFactor)));
        momentum_changes = [];
        sinceInterval = currTime;
    }
}
```

Since the pressure is only updated once for every interval it appears less seamlessly than the volume data. For this reason its value jumps abruptly and its changes lag a little behind the changes in volume.



Here the y axis represents pressure scaled to the magnitude of the volume, while the x axis is time.

Introduction

Description:

Statistical concepts can be difficult to grasp due to their abstract nature. These concepts become much easier to grasp when they can be seen in relation to the systems with which they correspond. This simulation was created with that in mind, by interactively modeling an ideal gas system and displaying its statistics alongside the user is able to see visually how the statistics emerge as the system evolves.

A JavaScript program is used to create a box in which a number of particles are set to collide elastically. This simulates the properties of an ideal gas. Data is collected from the system and displayed in graphs. An input box appears underneath the display through which the user is able to specify parameters. Just above the input box, a slider allows the user to compress the box's volume using a partition.

Development:

The project began with a number of prototypes designed around a similar model using different programming languages. For the sake of user accessibility Javascript was chosen and the final version was designed embedded within a web-page with HTML forms acting as the user interface.

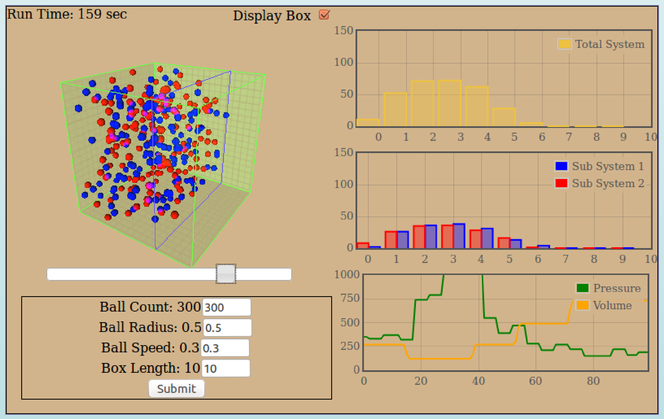
Graphical Display:

WebGl was used for the visual display of the final simulation. It utilizes specialized chunks of code called shaders which are sent to the user's graphics card and carry out specialized calculations having to do with the display. These include the coloring, texturing, and lighting of the scene as well as the blending effect which gives the box its transparent appearance.

Data Plots:

Flot is a JQuery plugin which was used to create the graphs which represent data in this simulation real time. It consists of a few simple functions which run along side the main program and take in preformatted data arrays which they display as graphs.

For more on WebGl visit http://www.khronos.org/webgl/
For more on Flot visit http://www.flotcharts.org/



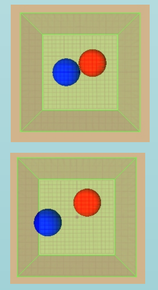
"WebGL and the WebGL logo are trademarks of the Khronos Group Inc."

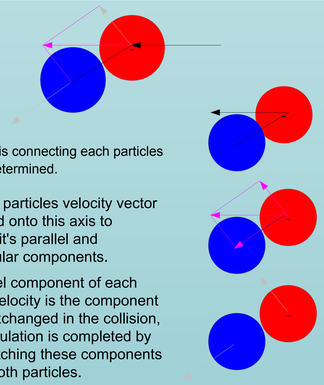
Methods

Collision Detection

Basic Elastic Collision:

Whenever two particles collide a simple vector calculation is performed. By projecting the particles velocities onto the axis of collision the obstructed component of both particles velocities can be exchanged. This calculation ensures that momentum is exchanged so that the particles will react to collisions properly even when they hit at an angle.





First the axis connecting each particles center is determined.

Then each particles velocity vector is projected onto this axis to determine it's parallel and perpendicular components.

The parallel component of each particle's velocity is the component which is exchanged in the collision, so the calculation is completed by simply switching these components between both particles.

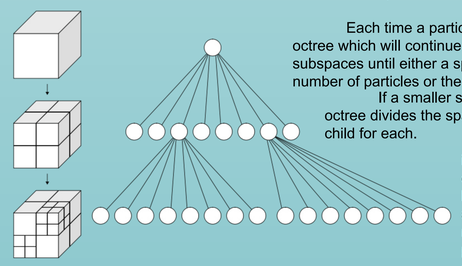
Optimization with Octree Partitioning:

During the simulation particles are only able to collide with one another when they are in close proximity. This means that we need only check for potential collisions between nearby particles as those that are far away have no chance of colliding.

In order to optimize the potential collision search the space is recursively divided into 8 smaller subspaces using an octree.

Whenever a ball moves it is added to the octree and sorted into a subspace. When the collision search takes place particles within the same subspace are retrieved as pairs and checked for mutual collisions.

This spatial partitioning effectively reduces the search time for collisions since the potential collision pairs which it generates are a small subset of the whole system of particles.



Each time a particles position changes it is added into the octree which will continue to sort the particle into smaller and smaller subspaces until either a space is found with less than a specified number of particles or the trees maximum depth is reached.

If a smaller subspace is called for but none exists then the octree divides the space into 8 smaller subspaces and creates a child for each.

During the collision search the octree is consulted for potential particle-particle co0llisions and potential particle-wall collision. It finds these by checking the contents of the smallest subspaces it has generated and returning the particle-particle and particle-wall pairs. These pairs are then checked for collisions.

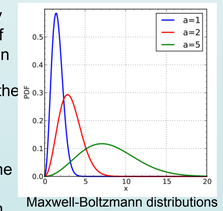
Program can be found live at **http://www.csmarlboro.org/cdaniels/idealgas**
Source code available at **https://github.com/cdaniels/Collision-Simulation**

Results

Maxwell-Boltzmann Distribution

velocity distribution across system

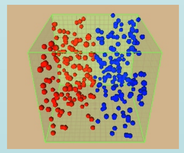
The Maxwell-Boltzmann distribution is a statistical distribution of velocities which is closely associated with Ideal gases and as such is one of the properties of an ideal gas which this simulation aims to reproduce.



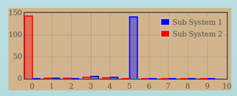
Maxwell-Boltzmann distributions

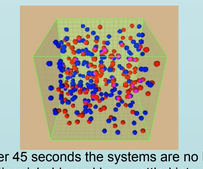
The distribution appears because of how the collisions which take place during the simulation redistribute momentum across the system. This leaves us with a situation where the median velocities are more likely to occur than the extreme ones. If the number of particles within a certain velocity range is plotted against velocity a smooth curve should appear as the simulation runs.

In order to demonstrate how velocities redistribute in our system, the particles are divided into two subsystems, one **red**, and one **blue**. The **red** system begins motionless, while the **blue** begins at the specified velocity. After about 45 seconds collisions have bled the two systems together and both have velocities which approach the Maxwell-Boltzmann distribution.

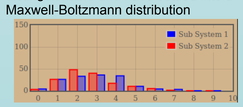


Initially the sub systems begin with different velocities





After 45 seconds the systems are no longer distinguishable and have settled into a Maxwell-Boltzmann distribution



Ideal Gas Law

Pressure Volume relation

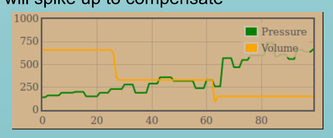
The Ideal Gas Law is an equation which relates the pressure, volume, and temperature of an ideal gas system. In its real world context the pressure times the volume is found to be proportional to the temperature by the number of particles in the system times a constant.

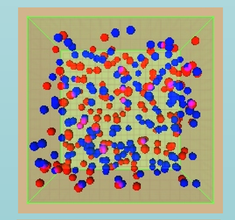
$$PV = nRT$$

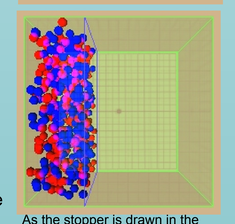
In this simulation however, all of the collisions which take place are completely elastic and the particles all remain in the box once the simulation has begun. This means that the temperature and number of particles remain constant and we are left with an equation which describes the pressure and volume of the system as inversely proportional to one another.

$$nRT = K \qquad P = K\frac{1}{V}$$

This is exactly what appears in the simulation, if the volume is decreased by dragging the partition slider then the pressure will spike up to compensate







As the stopper is drawn in the enclosed volume shrinks