Picturing Equilibrium

Wesley Tung

December 10, 2015

1 Abstract

Every chemical *reaction* combines component *reactants* to form resulting *products*. In certain cases, and under certain quantities of reactants, the amount of products and reactants being made is in equilibrium. This means that is the rate at which products are being made equals the rate at whcih reactants are being made.

This project demonstrates the effects of chemical equilibrium and explaining the reasoning and conditions behind it. This will be accomplished by creating a model using VPython and pairing them with explanations.

2 How I did it

2.1 Analysis of Code

I started by analyzing Bruce Sherwood's VPython example "gas.py". I had no experience whatsoever, so I had to look at every single line of code in order to decipher each line's meaning and contribution to the module. I also had to pay special attention to how he modeled collisions becuase I knew that collisions would bcome a vital component to modeling the creation of compounds. After four days of annotating, I was able to understand the meaning of Bruce's code.

2.2 Easy Modifications

The next step was for me to start modifying the code. Bruce's code only had on type of atom in it, so I had to rewrite it so that the model contained two atoms of different size and color since atoms of different elements were of different sizes and I want to color code my atoms.

I noticed that Bruce created atoms by appending spheres to a list with a random position, a set radius, and a color. He then created seperate lists of all the atoms' position, momentum, radius, and masses. He then ran those same lines of code using loop to create how many atoms he wanted. I deduded, that if I wanted to create more atoms, I would have to do the same exact thing, but change the information such as color and radius to achieve a different kind of atom.

So I duplicated his code and looped it the same amount of times as the other atom. I made this atom yellow and signiifcantly smaller than the other. I designated this atom as Hydrogen and the original was Iodine.

After creating Hydrogen and Iodine, I combined all the seperate lists of position, momentum, mass and radius together into combined lists of position, mass...etc. Then I created arrays of those combined lists. The calculations for collisions that occured later in the code required array mathematics so I needed to create arrays of my lists.

2.3 Harder Modifications

The next thing I had to do was to define functions dictating how the atoms would react when hit one atom hits another. In order to do this I needed to change the function and the mathematics so that When two atoms or the same type hit each other, they bounce off. And when atoms of different types hit each other, they needed to stick together for a certain amount of time, than split apart again.

I studied how Bruce modelled his original collisions, and I worked through his mathematics. The original program only had atoms bouncing off of each other and it used a lot of physics. It started off by defining a value for mass, velocity and radius for each of the atoms involved in the collision.

It then defined a position function for each atom that dictated the positions of the atoms at the time of impact.

$$pos[i] = pos[i] - \frac{momentum[i]}{m[i]} * deltat$$

Afterwards, we had to transform the momentums of each of the atoms to a center of momentum frame so we could modify the direction of the collided atoms.

$$pcm[i] = p[i] - (totalmomentum) * \frac{m[i]}{combinedmass}$$

Through another series of equations, we defined a unit vector of the two position vectors of the atoms combined, and then we reversed the direction of the atoms so that that they bounced in the opposite direction then thier original trajectories.

$$rrel = norm(pos[j] + pos[i])$$

$$pcm[i] = pcm[i] - 2 * dot(pcm[i], rrel) * rrel$$

We then redefined the momentum of the atoms usingthe new center of momentum.

$$p[i] = pcm[i] + ptot * \frac{m[i]}{mtot}$$

And finally, we redefined our position function so that it incorperates the new direction into the new positions of the atoms over time.

$$pos[i] = pos[i] + \frac{momentum[i]}{m[i]]} * deltat$$

This code was duplicated so that the second atom would exhibit the same properties. When this code is run, the I decided that I should keep this code and have the program execute this when it detects two atoms of the same radius colliding. However, if two atoms were of different length, I needed the atoms to "stick" together and have the same momentum and direction.

I accomplished this by duplicating the above code but changing the one of the equation for the second atom from:

$$pcm[j] = p[j] - (totalmomentum) * \frac{m[j]}{combinedmass}$$

to

$$pcmi = pcmj$$

This way, the direction and momentum of the second atom will always be the same as the first atom. I also took out anything related to reversing direction.

2.4 Hardest Modifications

The last two and hardest modifications for me were the mouse event and seperation of molecules after sticking together for a certain amount of time.

I first worked on the seperation of atoms function. I utilized class oriented code in order to make this work. I created a class called "Collided" which consisted of the first atom, the second atom, the time at which they collided, the momentum of the two atoms, and the position of the two atoms. Every single time that the function ran into two atoms of differing radii that collided, it first checked to see whether those two atoms were already in a collision by checking a list of all atoms that were currently collided.

If that search came up a matching pair of atoms, than that pair of atoms was already in the list, and we instructed the function to continue to the next part of code. If that search didn't come up with a matching pair, we instructed the function (using the math above) to give the atoms the same momentum and direction and create an object called "compound" with with of the atoms, the time at that moment, the momentum and the position of those atoms.

After the object had been created, I wrote a function that continuously checked the time off each instance of "Collided" against the current time. If the current time - the time off collision was greater than three, I executed a function that caused the atoms to seperate from each other. Otherwise, the function would do nothing and continue on. It took me quite a long time to figure this out, and I had to have outside aid to help me with this.

The next thing I had to accomplsh was the mouse event that modified a variable of equilbrium. I decided to modify the amount of reactants in the equilibrium system by adding more.

The first thing I had to do was create the function that would respond to the mouse click. I had some more outside help for this because I thought I would have to do something fairly complex, but I figured out int he end that I just needed to reutilize the function that appended atoms into a list.

In the end, I rewrote that block of code so that it would add fifteen more atoms of each type into the animation. I then bound that function to a mouse click. I also had to make a lot of variables global so that they would translate into the other parts of the function.

3 How To Operate the Animation

Operating my function is acutally fairly simple. When activiating it, you will be given a small rectangular screen with a box containing thirty atoms total. Fifteen atoms a hydrogen and fifteen atoms iodine are present. Simply click on the animation to add fifteen more atoms to the animation. The atoms will start with a random position and velocity but will quickly combine with other atoms to acheive equilbrium.

4 Future Improvement

Currently, the animation only depicts one stressor of equilibrium. To improve the project, I could create more animations that depict the other stressors of equilibrium like temperature or volume. In addition, I could improve this specific animation by creating an option to take away atoms, or create a graph that tracks the amount of atoms that are combined in a compound at any given time. Also, I'm not quite sure that my code for seperation is working properly, so I could also improve that.

5 Sources

Bruce Sherwood's VPython example "gas.py"

Special thanks to all the mentors for guiding me along in the process! And special thanks to Nathan Walters for helping me out when I was struggling with objects!