Final Project Write-up

Kyle D. Patterson

May 2018

I built a 2D air-molecule-based simulation to model heat transfer in a computer case with the goals of testing the effect on the CPU temperature over time for different case volumes, case shapes, numbers of fans, fan directions, air molecule density, and rate of heat generation. My code is broken down into simulations of increasing complexity to model different physical phenomenon including, collisions, thermodynamics, pressure, air currents and equilibrium points. First, I start with two air molecules colliding elastically and expand to a boxed area with a CPU that produces heat at a constant rate. I measure the temperature of the CPU over time and compare across different case and fan setups. I use the term "particle" generally to describe material entities which interact with each other and can exchange heat in the form of kinetic energy. I model composite air molecules which each represent multiple moles, the CPU itself which is modeled as a single object (but retains its temperature better than air molecules in collisions because of its larger area), the walls of the computer case (which are conductive but immobile), and the fans which are non-conductive but affect the velocity of the particles.

1 Introduction

Motivation Moore's Law has existed for decades, allowing for unprecedented rates of increases in computation power. However, recent gains have been increasingly limited by the laws of physics, namely electro-magnetic interference and heat production. More efficiently understanding the heat production and transfer in both consumer and industrial computer systems is critical to optimizing these systems for maximum performance.

The System Assume that I will model an area of a case with only a rectangular region, of uniformly distributed temperature, as the heat spreader (piece of metal that distributes heat) of the CPU. I will assume that the CPU draws a constant wattage and therefore generates heat at a constant rate. I will also model an area outside the case. In the open areas (i.e. areas without a wall of the computer case, the CPU heat spreader, or other air molecules), I will randomly generate a specific quantity of air molecules which represent the average molecule of air. The air molecules will initially have an ambient amount of kinetic energy corresponding to room temperature and be travelling in random directions.

Heat Transfer As the CPU runs, it will generate heat, thereby raising its temperature by producing heat. Heat is a measure of energy transfer with units of joules. When heat is transferred from a solid metal to molecules of air (e.g. H2O, N2, O2), the energy transfer can be in potential or kinetic energy. I will assume that these molecules do not change phase or react and that they only exchange kinetic energy. This is a reasonable assumption since with the exception of water vapor, which is not a large part of air, most gases do not change phase around room temperature. Thus, the molecules will transfer kinetic energy (according to Newton's Third Law of equal and opposite reactions) when they bump into one another. The molecules will additionally change speed accordingly (kinetic energy

$$\mathrm{KE} = \frac{1}{2} m v^2$$

with mass m and speed v). Since I am only modeling change of kinetic energy, these collisions must be elastic, meaning kinetic energy is conserved. Although some collisions in a gas are not elastic, others are super elastic meaning that they gain energy, so on average the collisions are elastic. So, elasticity is a reasonable assumption. After bumping into one another, the air molecules in my system will change velocity but conserve both kinetic energy and momentum. Thus, where u_1 and

u_2 are the velocity vectors before, m_1 and m_2 are the masses, and v_1 and v_2 are the velocity vectors after.

$$m_1u_1 + m_2u_2 = m_1v_1 + m_2v_2 \qquad \text{(Conservation of momentum)}$$

$$\frac{m_1u_1^2}{2} + \frac{m_2u_2^2}{2} = \frac{m_1v_1^2}{2} + \frac{m_2v_2^2}{2} \qquad \text{(Conservation of energy)}$$

$$\Rightarrow v_1 = \frac{u_1(m_1 - m_2) + 2m_2u_2}{m_1 + m_2} \text{ and } v_2 = \frac{u_2(m_2 - m_1) + 2m_1u_1}{m_1 + m_2}$$

The air molecule Different kinds of molecules have different specific heats and are different sizes, but air is locally (on the scale of centimeters) very similar in composition, i.e. similar in the distribution of different types of molecules measured from two different samples close together. So, I will make the simplifying assumption that a composite molecule will effectively mirror the heat-transfer phenomena of a more complex system with multiple kinds of molecules.

The CPU The final goal of this simulation will be to do thermodynamics experiments, tracking the temperature of the CPU, with different case volumes, case shapes, numbers of fans, fan directions, air molecule density, and rate of heat generation. I will simplify my calculation of temperature by using kinetic energy divided by the size (so in this 2D simulation area, I'll use the assumption that size is proportional to thermal mass) of each particle. Thus,

$$T = \frac{\text{KE}}{A}$$

The CPU will generate heat at a constant rate and the change in energy of the molecules of the heat spreader will be equal to this amount. When air molecules pass through the rectangular region in the case where the heat spreader is, I will calculate a weighted average of temperatures between the heat spreader and the molecule (based on an experimental parameter for how much bigger the heat spreader should count for) and then redistribute heat. For example, with particles with areas A_1 and A_2,

$$T_{\text{avg}} = \frac{m_1\text{KE}_1 + A_2\text{KE}_2}{A_1 + A_2}$$

Then, I will take the differences in temperature of between each particle and the average temperature of the two particles

$$\Delta T = T_{\text{avg}} - T_{\text{particle}}$$

Now, the new temperature of each particle after the collision is equal to the old temperature plus some fraction of T_{avg}. The fraction of T_{avg} is smaller when the average thermal conductivity (β) of the particles is less.

$$T_{\text{new}} = T_{\text{old}} + \Delta T \cdot \beta$$

Furthermore, since I am simulating with the assumption that energy will only be transferred in the form of kinetic energy, I can relate the heat generation, not removed by air molecules by

$$\text{KE}_{\text{avg}} = \frac{3}{2}kT$$

where k is Boltzmann's constant ($1.38 \cdot 10^{-23}$ joules per kelvin).

2 Code

The simulation consists of a step function; a two dimensional array to store the positions of case edges, the CPU, and the air molecules; and a class to create air molecule objects with the properties of position, velocity, acceleration, and kinetic energy. I will express output using a graph of CPU temperature, a GIF of air molecule movements, and a plot of pressure inside and outside of the case.

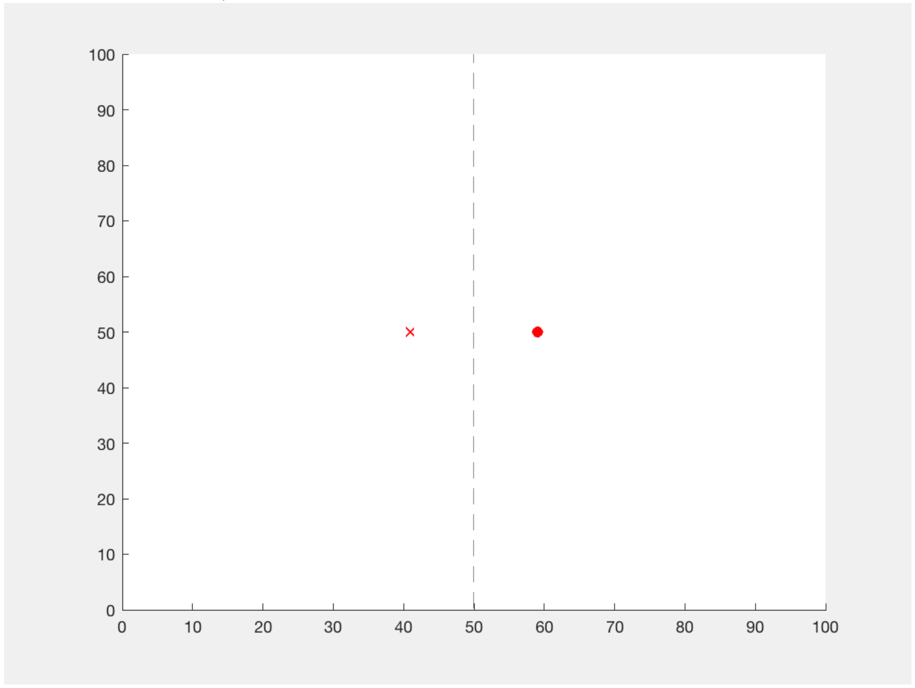
Air_molecule class Basic class for the simulation. Holds position, velocity, kinetic energy, ambient kinetic energy, mobility, mass, whether collisions are enabled for that air molecule, the specific marker to use when plotting that air molecule, and the index of that air molecule in the 'air molecules' array used in the simulation. Each air molecule has a mass of 1 and a default kinetic energy of 100.

Case_molecule class Subclass of air molecule. I chose to subclass Case_molecule from air molecule since the edges of the computer case have positions, kinetic energies, dimensions, masses, and ability to transfer energy/interact with each air molecule much in the way that a air molecule would interact with another air molecule. A future simulation might incorporate more of the interaction functions in air molecule for Case_molecule objects. This class is immobile (and hence always 0 velocity) and can have any mass. The walls of the PC case for instance can collide with and transfer heat with air molecules (air molecule objects).

Heatspreader class Subclass of air molecule. This class is like a air molecule except that it has a rate of heat generation property and a method that increases its kinetic energy every time step based on that rate.

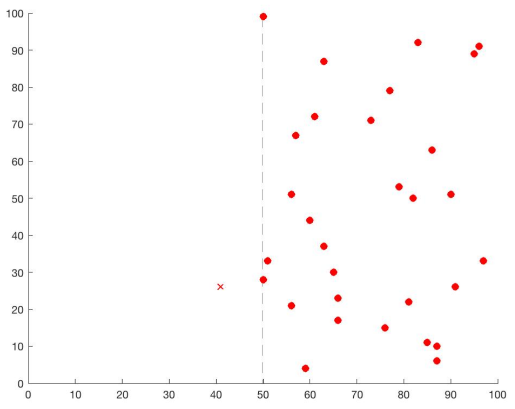
3 Exercises

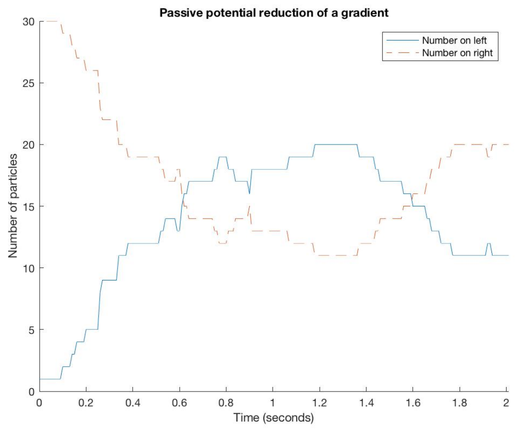
Part 1: Collision of two molecules of equal mass and velocity heading toward each other
Note: The x's represent air molecules starting in one region and o's represent air molecules starting in another. The shape has no other effect than to distinguish the two. All of my simulations use time steps of 0.01 second. In some simulations, the air molecules have different amounts of energy. Redder air molecules represent higher kinetic energy and yellower air molecules represent lower energy. I created two air molecules approximately centered in each half of a 100 by 100 plot, which I call the 'canvas', and set the velocities to point toward the center of the 'canvas'. I simulated using a time step of 0.01. The air molecules travel inward and collide with each other upon reaching the center then bounce back outward with equal but opposite velocities. With a larger time step like 0.1, the air molecules still collide but there is some space between them when they do. This is because the new position of one air molecule will be in the same space as the other air molecule, so the simulation calls the collide function. This is fixed using a recursive call of the 'move_helper' function to evaluate every point along the path. If the molecules reach the edge of the simulation then they bounce back inward (implemented with reflection of velocity across normal to edge, i.e. perpendicular velocity). See the 'part1.gif' file.



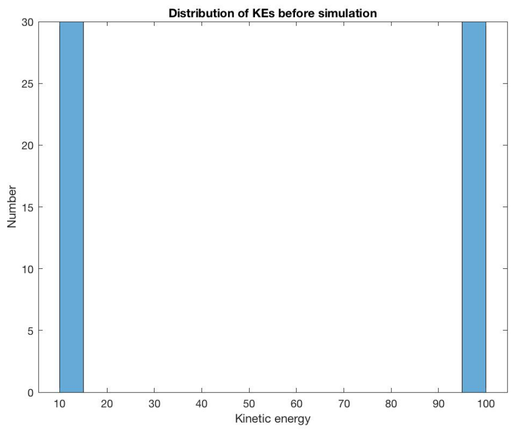
Part 2: Collision of two molecules of unequal velocity but equal mass This simulation is similar to Part 1 except now the air molecules have different kinetic energies and thus travel at different speeds. They collide slightly to the left of the dividing line, as the air molecule on the right starts with more energy and therefore travels faster initially. The colors of the air molecules range from yellow to red with increasing kinetic energy. After the collision, both air molecules have equal amounts of energy. If the molecules reach the edge of the simulation then they bounce back inward (implemented with reflection of velocity across normal to edge, i.e. perpendicular velocity). See the 'part2.gif' file.

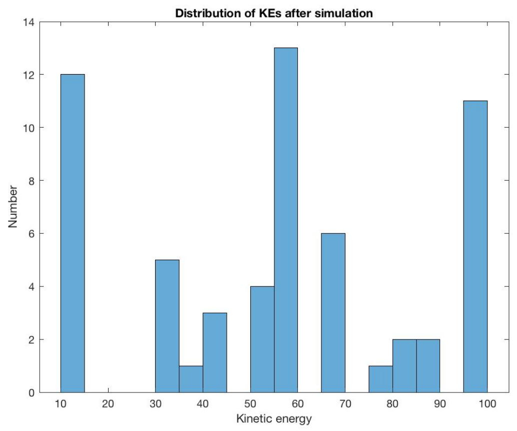
Part 3: Box with higher density of molecules on one side In this simulation, I put 30 air molecules in random starting positions on the left side and only one air molecule on the right. The air molecules initially start travelling in random directions but with equal velocities. After 2.0 seconds of simulation time (not real-time per se), the gradient has almost entirely disappeared and there are approximately the same number of air molecules on either side meaning the gradient had mostly disappeared. The molecules start at the same temperature and therefore there isn't any transfer of energy. If the molecules reach the edge of the simulation then they bounce back inward (implemented with reflection of velocity across normal to edge, i.e. perpendicular velocity). See 'part3.gif'.



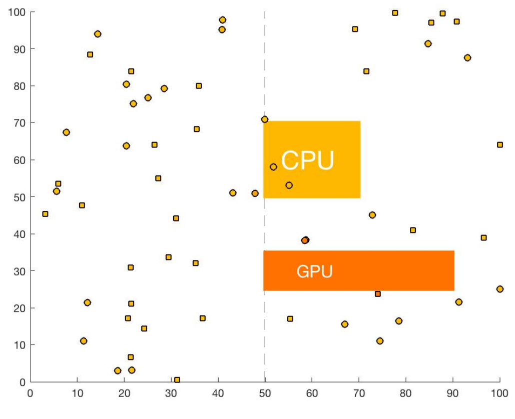


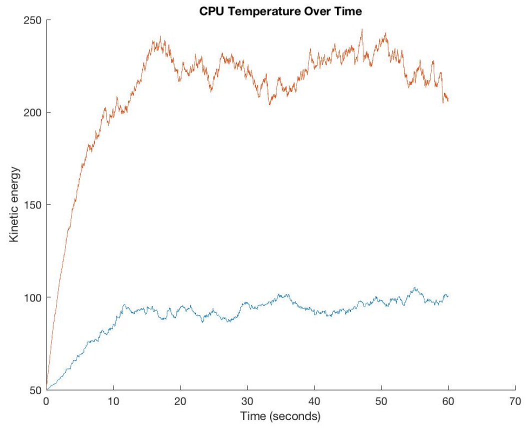
Part 4: Box with higher temperature molecules on one side This simulation was similar to the previous one in that I setup an initial gradient except this time with differing kinetic energies. I put 30 air molecules starting on either side. However, I gave the air molecules on the right kinetic energies of 100 and the air molecules on the left kinetic energies of only 10. After running the simulation for 2.0 seconds with a time step of 0.01, the distribution of kinetic energies was still bi-modal, but the peaks had decreased in height significantly (to less than half of what they were previously) and a number of air molecules had energies between the peaks, meaning that the KE gradient had mainly disappeared. See the histograms below. The distribution tends to be more evenly distributed on the right since the higher kinetic energy air molecules are biased to collide more often by virtue of their higher speed. If the molecules reach the edge of the simulation then they bounce back inward (implemented with reflection of velocity across normal to edge, i.e. perpendicular velocity). See 'part4.gif'.

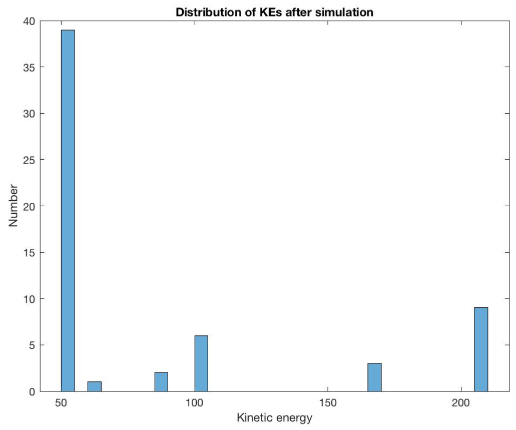


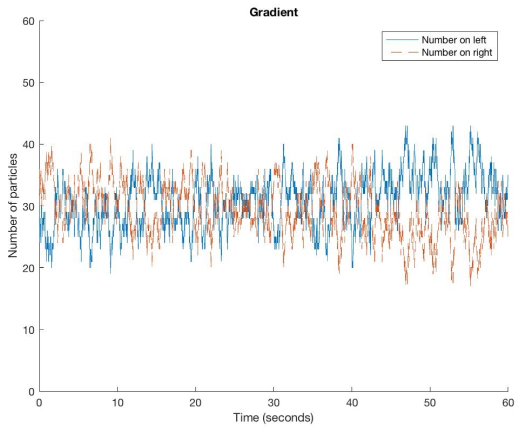


Part 5: Different numbers of air molecules with CPU heat spreader without a case to see how purely air density effects cooling *Note: The squares represent air molecules starting in one region and 'o's represent air molecules starting in another. The shape has no other effect than to distinguish the two. If a molecule reaches the edge of the simulation then they are replaced during the next time step with a molecule on one of the edges of the simulation traveling at an angle* $\pm(\frac{\pi}{2}-\alpha)$ *radians of the normal to that edge, where* α *is a simulation parameter located in the constant properties of the Air_molecule class.* This simulation adds a rectangular region for the CPU that the air molecules do not collide into but can instead pass over. When an air molecule enters this region, it transfers some energy with the heat spreader according to the differences in temperatures between it and the CPU. I simulated a system with only a CPU and another system with a CPU with a GPU. Both systems had the same long term behavior; over time, temperature leveled off to an equilibrium point. This likely happened as the molecules of air removed more heat energy as the difference in temperatures between them and the CPU/GPU increased. See the figures below. The first figure shows the CPU temperature over 60 seconds with a rate of heat generation of 10 KE per second, an initial temp of 50, a CPU size of (20, 20), and a CPU bottom left corner position of (50, 50) on a 100 by 100 canvas with 15 particles initially starting on the right and 15 particles initially starting on the left, each with a thermal conductivity of 0.5 and temperature of 50. Over time, the temperature levels off as the CPU reaches its equilibrium point. The equilibrium temperature point likely depends on the rate of heat generation of the CPU and the GPU, the size of each, the number of air particles, and the thermal conductivities of the air particles and the heat spreaders.







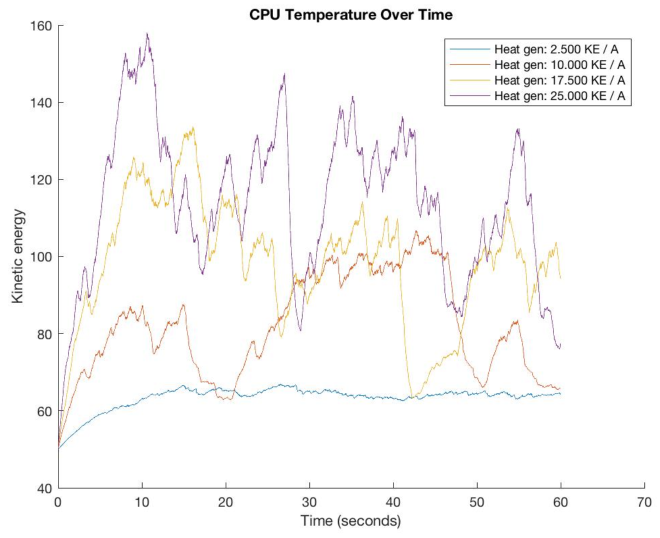


The graph of CPU Temperature Over Time shows the GPU temperature in red and CPU temperature in blue. The GPU rate of heat generation was 150 KE per second whereas the CPU rate of heat generation was only 10 KE per second. Both heatspreaders had the same area but the GPU was longer. I placed both heatspreaders near the center to reduce the possibility of particles spawning on the outside edge spawning directly on one of the devices. The graph of distribution of KEs after simulation shows peaks at 50 (the ambient KE that new particles spawn with), 60 (perhaps particles that had touched other particles with slightly more kinetic energy), 75 and 100 (the points the equilibrium temperature of the CPU oscillated between), 160 (about 2/3 of the way between the equilibrium temperatures of the two heat spreaders, so maybe the temperature of particles that had touched both), and 225 (the equilibrium temperature of the GPU). Finally, the graph of gradient (particles on the left versus right) appears to show a slightly larger number of particles on the left as time goes on, while the principle of entropy would suggest an approximately equal number on both sides (as in Part 3). This may be because the CPU and GPU were on the right so particles on this side were likely to be heated above ambient temperature and therefore travel faster. Because they were travelling faster, these particles would be likely to spend less time on the right side of the simulation. See the video of this simulation, rendered for 60 seconds of simulation time. The paths of the particles are easier to follow when the video is played back in 0.25 speed. Over time, we can see the effects of the CPU and GPU heating up but the CPU staying at a much lower equilibrium temperature.

Part 5a Hypothesis 1: Regardless of their rate of heat generation, after some amount of time any CPU will reach a steady state for temperature.

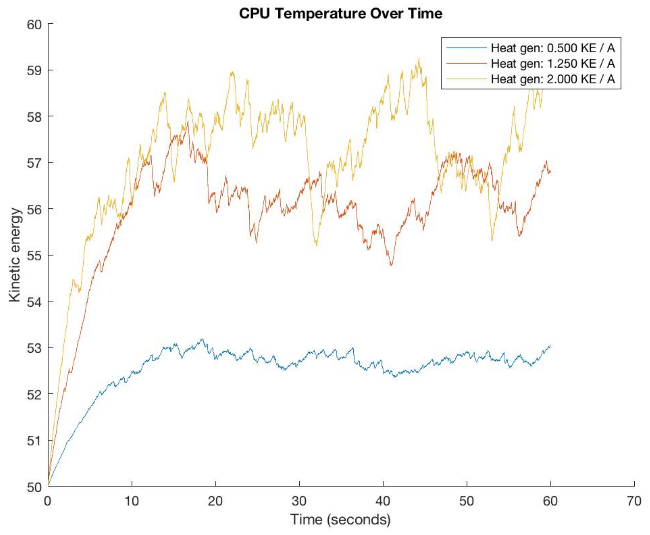
I made this hypothesis since the more energy the CPU heat spreader has, the more energy will be removed per collision with air molecules. Therefore, there should be some balance point. I used the following parameters with this simulation:

1. 60 air molecules
2. air molecule width = 2
3. air molecule height = 2
4. ambient kinetic energy = 50
5. thermal conductivity of air molecules = 0.5
6. thermal conductivity of CPU = 1.0
7. CPU width = 20
8. CPU height = 20



I found the following results: my data did not provide evidence for my hypothesis. The above plot shows the simulation run for different rates of heat generation (measured in KE of the entire CPU) per unit area (found by dividing by CPU area of $(20)(20) = 400$). While at a rate of heat gen 2.5 KE/A, the CPU temperature reached a steady state, higher rates of heat generation showed oscillating temperatures, with larger peaks and dips corresponding to higher rates of heat generation. The data supports my hypothesis in that none of the CPU simulations show a continuous increase in temperature, but the peaks are a phenomenon I did not expect. This behavior may have happened because of the heat removed by each particle being greater when the total temperature difference between the air molecules and the CPU was greater; thus higher rates of heat generation would lead to more jagged curves.

However, I predict that lower rates of heat generation might support or higher numbers of air molecules might support my hypothesis better by leading to less jagged curves. The results were more promising, as the curves appeared less jagged and all rapidly increased, but reached approximately different levels of equilibrium temperatures. This data supported my hypothesis.

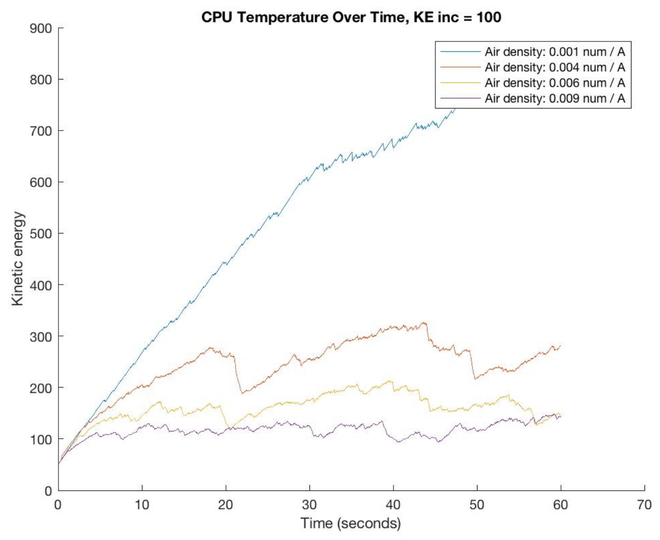


I tested the same parameters but with lower rates of heat generation.

Part 5b Hypothesis 2: Higher density of air molecules will lead to a more gradual slope of CPU temperature increase.

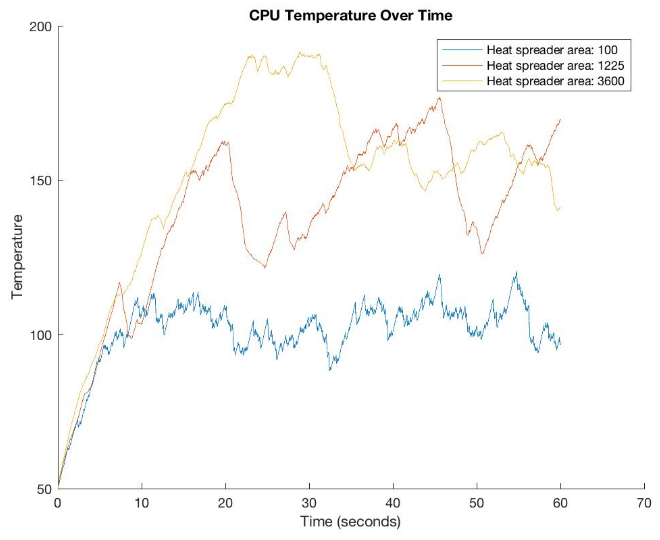
Hypothesis 3: Higher density of air will lead to lower equilibrium temperatures of CPU.

I used the same parameters as Part 5a except that I tested using a rate of heat generation of 10 KE / second (there is a mistake in the title of the below graph for KE inc). The higher the air density, the faster the rate of increase. With the exception of the lowest air density, 0.001 air molecules / area, the other curves plateau and reach equilibrium states. It's likely that the curve of CPU temperature for lowest density of air will also plateau eventually but the equilibrium temperature will be much higher since the number of air particles provides less cooling. Therefore, the data supported both hypotheses 2 and 3.



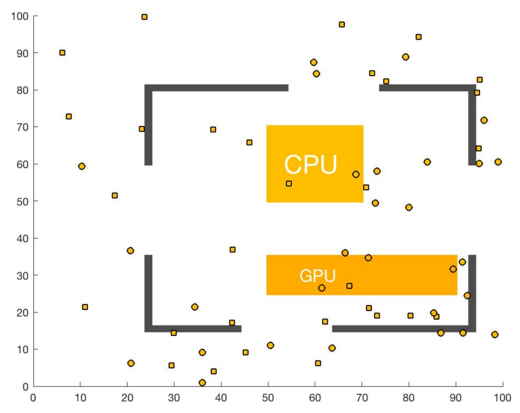
Part 5c Hypothesis 4: Larger heat spreader area will lead to greater CPU cooling and therefore lower equilibrium temperature.

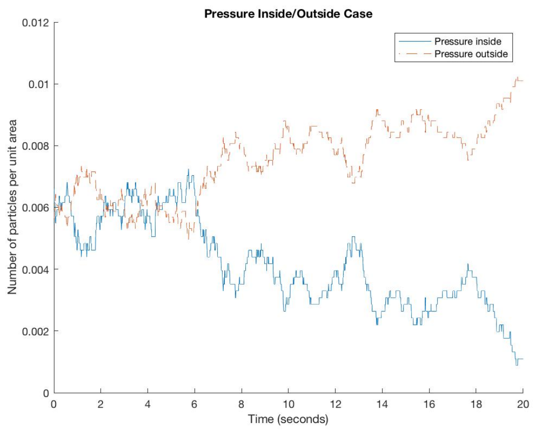
I used the same parameters as in Part 5b except I varied the length and height of the CPU heat spreader as the sides of a square with areas of 100, 1225, and 3000. The larger the heat spreader, the greater the rate of cooling and equilibrium temperature. The smaller curves provided more jagged curves as fewer particles hit them and so the rate of change of heat was more discrete.



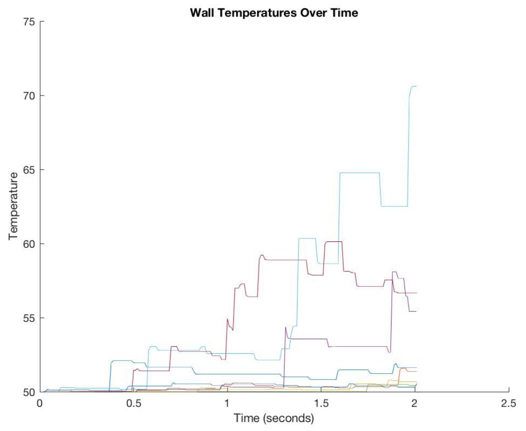
Part 6: Test the above exercise with a case that is not thermally conductive In the real world, examples of non-thermally conductive computer cases can be found in various plastics used for computer cases. I added a case which impedes the movement of particles. The case was not thermally conductive but had holes in it by which particles could enter. I tracked the pressures inside out and outside the case by counting the particles divided by each area. Pressure increased outside and decreased inside as time went on, probably because of the hotter components warming the air and causing it to travel faster inside the case. I was not able to introduce the walls without error as after some amount of time, particles began getting stuck on the left and bottom edges of

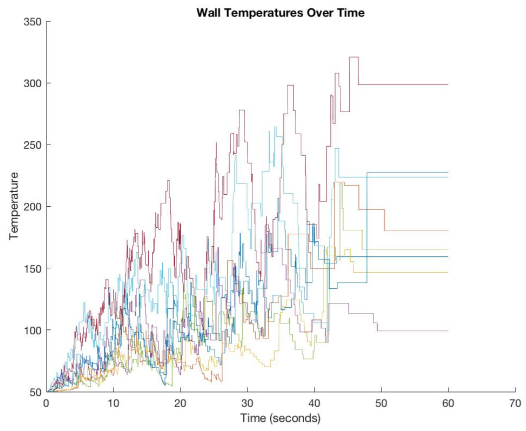
the simulation, near the entrances to the case and tangent to the left or bottom axis. This resulted in a wonky pressure graph in which over time, fewer particles are able to enter the case.





Part 7: Making the case thermally conductive An example of a thermally conductive case is found in any of the later models of Apple and HP laptops, which use all or partial aluminum cases. I made the case thermally conductive and found a strange increasing step pattern with the walls. See the video of this simulation, rendered for 60 seconds of simulation time. The parameters for this video are in the description section of the video. Similarly, I was not able to do hypothesis testing on Part 7 as I had the same issue as in Part 6. This lead to a sustained increase in the temperatures of the CPU, GPU, and the walls. Interestingly enough, the increase in the walls was step like, likely because the number of particles was small enough that each wall didn't change temperature too often. The walls showed similar growing oscillations to the CPU at higher temperatures.





Validation For code validation, I have investigated edge/boundary cases in addition to testing functions like collision and heat redistribution in the above exercises.

Tests

1. Movement function/collisions: I tested these functions primarily with the first simulation which shows the collision of two air molecules. I also checked the make sure that the spaces the particles had left were marked empty in the grid by inspecting the grid after simulations. There is some strange movement of particles occasionally which is caused by each particle taking its move turn in sequence (so particles show the effects of colliding with where other particles usually are). The effect of this is small for small time steps.

2. Energy exchange function: I tested whether this averaged the energies of two particles correctly (thermal conductivity of 1 for both) then whether the weighted average was done correctly.

3. CPU temperature: I first tested the CPU heat increase without any effect of air molecules. Then, I added one air molecule alone, with bouncy edges of the simulation so the air molecule's

energy wouldn't be reset, and plotted temperature to see how temperature would change. Then, once I was satisfied that with the plot showing the repeated averaging of an increasing temperature and a constant temperature, I added more particles and examined this graph for similar features.

4. Particles movement/overall trends: The visualizations were especially helpful for this since they display the key parts: size, position, and temperature/kinetic energy of the parts of the simulation and I was able to verify the behavior of many of my functions simply by watching the collision of two particles or change in color on my plot.

Boundary conditions

1. *Is it possible for air molecules to spawn on top of each other? If so, do they get stuck or what happens?*

   After choosing a random position for a air molecule, the code checks that position in a grid system that keeps track of whether each coordinate is empty, so this is guaranteed to never happen.

2. *What happens when a air molecule reaches the edge of the simulation?*

   In the Parts 1, 2, and 3, the air molecules bounce off the edges. In later parts, the air molecules are allowed to pass out of the simulation and are simultaneously replaced by air molecules with ambient amounts of kinetic energy and default properties entering along a random edge in a direction between $\pm(\frac{\pi}{2} - \alpha)$ of the normal to that edge.

3. *Do air molecules ever get stuck or have glitchy behavior?*

   (a) air molecules can get stuck in a corner of the simulation in Parts 1, 2, and 3 when the air molecules bounce of the edges. This likely happens because of conservation of velocity. If the air molecules initially spawn so that they are in particular positions, their collisions can cause them to repeat their paths in a cyclical pattern. This happens to a more extreme scale in the corners.

   (b) air molecules are guaranteed not to skip over each other (in parts 5 and later) since my movement function checks over each cell in the grid system when a particle is tracing a given path. This is more computationally intensive than the methods I use in earlier parts which only check the new predicted position of the particle but produces far more accurate trajectories and satisfying collisions when the paths of particles appear to cross (not having a collision when collisions are enabled isn't very satisfying).