IDEAL GAS SIMULATION

Advaith Sethuraman
Anup Hiremath
Humdaan Mustafa
Period 4

DESCRIPTION

- This program takes into account the temperature, pressure, volume, velocity, and number of molecules and uses the Ideal Gas Laws along with Maxwell-Boltzmann distributions to assign the appropriate values to the particles
- The animation itself consists of 12 discrete sequences that alternate in changing temperature, number of particles, volume, and pressure while keeping the dependent variables constant.
- As a result, we change one variable at a time and see the effect of the change on the pressure of the gas, which we can observe as collisions per second on the screen.

ALGORITHM SUMMARY

1. Color Algorithms
2. Velocity Algorithms
3. Ideal Gas (V & T) Algorithm
4. Ideal Gas (P & V) Algorithm
5. Random Velocity Algorithm
6. Bounce Back Algorithm
7. Volume Change Animation

COLOR ALGORITHMS

- A useful feature in showing relative temperature changes is to change the colors of the particles themselves. Our animation does this by taking temperature, linearizing the value to wavelength, and using a piecewise function to determine the color grade of the certain wavelength.
- This code is for the scaling of the wavelength to the r/g/b color values themselves. The values are multiplied by the max intensity and then raised to the power of the gamma factor.

```
double set_r(double temp){ //takes temperature as input
        double G= .75; //gamma is set to .80
        int MaxI = 255; //this is the maximum value
        double l = 13*(temp + rand()%7 - 3.5)+ 380; //linearization
        double red;
        double green;
        double blue;
        double scale; //factor
        if((l >= 380) && (l<440)){ //piecewise function
            red = (440-l) / (440 - 380) ;
            green = 0;
            blue = 1;
        }
        else if((l >= 440) && (l<490)){
            red = 0;
            green = (l-440) / (490 - 440) ;
            blue = 1;
        }
        else if((l >= 490) && (l<510)){
            red = 0;
            green = 255;
            blue = (510-l) / (510 - 490);
        }
        else if((l >= 510) && (l<580)){
            red = (l-510) / (580 - 510);
            green = 1;
            blue = 0;
        }
        else if((l >= 580) && (l<645)){
            red = 1;
            green = (645-l) / (645 - 580);
            blue = 0;
        }
        else if((l >= 645) && (l<781)){
            red = 1;
            green = 0;
            blue = 0;
        }
        else{
            red = 0;
            green = 0;
            blue = 0;
        }

        if((l >= 380) && (l<420)){
            scale = .3 + .7*(l - 380) / (420 - 380) ;
        }
        else if((l >= 420) && (l<701)){
            scale = 1;
        }
        else if((l >= 701) && (l<781)){
            scale = .3 + .7*(780 - l) / (780 - 700);
        }
        else{
            scale = 0;
        }

        if (red != 0){
            red = (MaxI * pow(red * scale, G)); //the value is multiplied by the max intensity then raised to the gamma
        }
        if (green != 0){
            green = (MaxI * pow(green * scale, G));
        }
        if (blue != 0){
            blue = (MaxI * pow(blue * scale, G));
        }

        return red; //returns red, the next two functions return green and blue
```

VELOCITY ALGORITHMS

- In real life, the velocities of gas particles follow a probability density function called the Maxwell-Boltzmann distribution, which is specified by the following formula:
- When you take the derivative of the distribution, you get a function for the x-value of velocity that has the highest probability for the particles to follow:

$$f(v) = 4\pi\left[\frac{m}{2\pi kT}\right]^{3/2} v^2 e^{-mv^2/2kT}$$

$$v_{rms} = \sqrt{\frac{3kT}{m}} = \sqrt{\frac{3RT}{M}}$$

m = molecular mass

M = molar mass

VELOCITY ALGORITHMS

- The root mean squared velocity is then used as the mean and the rest of the velocities are generated randomly around it, leading to a normal distribution if n (number of particles) is taken to infinity. Since we have a finite amount of particles, the approximation is somewhat crude but there exists a range of velocities nonetheless.

```
1105        particles[i].vx = rand()%(2*set_veloce(temp)) - set_veloce(temp);
1106        particles[i].vy = rand()%(2*set_veloce(temp)) - set_veloce(temp);
```

IDEAL GAS ALGORITHMS

```
1250        particles[i].vx = rand()%(2*set_veloce((px*px)/vol*temp)) - set_veloce((px*px)/vol*temp); //use ideal gas law to set velocity
1251        particles[i].vy = rand()%(2*set_veloce((px*px)/vol*temp)) - set_veloce((px*px)/vol*temp);
1252
1253        particles[i].circle_color.r = set_r((px*px)/vol*temp); // set color
1254        particles[i].circle_color.g = set_g((px*px)/vol*temp);
1255        particles[i].circle_color.b = set_b((px*px)/vol*temp);
1256
1257        if(particles[i].vx == 0){
1258          particles[i].vx = set_veloce((px*px)/vol*temp);
1259        }
1260        if(particles[i].vy == 0){
1261          particles[i].vy = set_veloce((px*px)/vol*temp);
1262        }
1263      }
```

```
1101    for(frame_num = 900; frame_num<1099; frame_num+=2){
1102
1103
1104      b_width = border_width + 200/198*(frame_num-900); //linearization of the border change
1105      vol = (px-b_width)*(pz-b_width);//volume calculation using cross sectional area
1106
1107      for(int i = 0; i < partnum; i++){
1108
1109       particles[i].circle_color.r = set_r(temp); //color assignment
1110       particles[i].circle_color.g = set_g(temp);
1111       particles[i].circle_color.b = set_b(temp);
1112
1113
1114      }
```

- The velocities of the particles are set proportionally to both the volume and the pressure. If pressure doubles, the temperature doubles and the velocity increases proportionally.

BOUNCE BACK ALGORITHM

```
597
598         for(int i = 0 ; i<partnum; i++){//bounce back algorithms
599             if(particles[i].cx+particles[i].vx>px-border_width-small_radius){//if it hits a border, the velocity (either x or y )
600              //is changed and the ball collides and bounces back
601                 particles[i].vx*=-1;
602                 particles[i].cx=px-border_width-small_radius;
603             }
604             else if(particles[i].cx+particles[i].vx<border_width+small_radius){
605                 particles[i].vx*=-1;
606                 particles[i].cx=border_width+small_radius;
607             }
608             else{
609                 particles[i].cx = particles[i].cx+particles[i].vx;
610             }
611
612             if(particles[i].cy+particles[i].vy>pz-border_width-small_radius){
613                 particles[i].vy*=-1;
614                 particles[i].cy=pz-border_width-small_radius;
615             }
616             else if(particles[i].cy+particles[i].vy<border_width+small_radius){
617                 particles[i].vy*=-1;
618                 particles[i].cy=border_width+small_radius;
619             }
620             else{
621                 particles[i].cy = particles[i].cy+particles[i].vy;
622             }
623
624             particles[i].radius = small_radius;
625
626             fill_circle(buffer, particles[i]);
627         }
```