Introduction

This final project I will be working on will be about rendering an incompressible fluid simulation. The goal of the project is to simulate and visualize the incompressible fluid movement within the project. An example, of something I want to simulate it the smoke from a fire, or the water vapor from our breathing activities. That being said, to keep in mind of what I am doing, the research paper that I have selected is not necessarily scientifically simulate the fluid movement or simulation, the idea is to simulate a sort of illusion for the user to feel like it is a fluid movement.

Related work

- GDC03.pdf (toronto.edu)

Description of algorithm

```
function IX(x, y) {
  x = constrain(x, 0, N - 1);
  y = constrain(y, 0, N - 1);
  return x + y * N;
}
```

The IX is a simple function to get the one-dimensional index within the 2-dimensional graphical space. The constraint serves to limit the possibilities of the value, x, and why not beyond the limit of the canvas size.

```
addDensity(x, y, amount) {
  let index = IX(x, y);
  this.density[index] += amount;
}
```

The addDensity function is a bit complex, it serves to add some “dye” within our canvas. To be more specific, the dye means how foggy it is. In short, the function serves to render how foggy the smoke should be.

```
addVelocity(x, y, mountX, mounty) {
  let index = IX(x, y);
  this.Vx[index] += mountX;
  this.Vy[index] += mounty;
}
```

The addVelocity function serves to add some velocity when rendering the fluid simulation. The function serves to render how fast the smoke should travel.

```
addfadeFluid() {
  for (let i = 0; i < this.density.length; i++) {
    let d = this.density[i];
    this.density[i] = constrain(d - 19, 0, 255);
  }
}
```

The addfadeFluid is a function serves to remove some of the smoke that had already been rendered. The function serves to stop the entire canvas from going white after the smoke is rendered for every pixel within the canvas, but also to display an illusion of 3-dimensional movement.

```
function diffuse(b, x, x0, diff, dt) {
  let a = dt * diff * (N - 2) * (N - 2);
  lin_solve(b, x, x0, a, 1 + 6 * a);
}
```

The diffuse function is responsible for the diffusion phenomena within the fluid simulation. One example of what a diffusion is, when someone is trying to drop a drip of coca-cola to a calm space river water, the water will have a sort of spread-out effect. That is what the diffusion really means. This function serves to render the spread-out effect of the fluid simulation.

```
function lin_solve(b, x, x0, a, c) {
  let cRecip = 1.0 / c;
  for (let k = 0; k < iter; k++) {
    for (let j = 1; j < N - 1; j++) {
      for (let i = 1; i < N - 1; i++) {
        x[IX(i, j)] =
          (x0[IX(i, j)] +
            a *
              (x[IX(i + 1, j)] +
                x[IX(i - 1, j)] +
                x[IX(i, j + 1)] +
                x[IX(i, j - 1)])) *
          cRecip;
      }
    }
    set_bnd(b, x);
  }
}
```

This is a linear solve solution, where you take a linear equation, and solve the linear equation for fluid simulation. The function returns value of the linear position over and over again, which help us to identify the spread-out pixel effect that we wanted when rendering the fluid simulation.

```
function project(velocX, velocY, p, div) {
  for (let j = 1; j < N - 1; j++) {
    for (let i = 1; i < N - 1; i++) {
      div[IX(i, j)] =
        (-0.5 *
          (velocX[IX(i + 1, j)] -
            velocX[IX(i - 1, j)] +
            velocY[IX(i, j + 1)] -
            velocY[IX(i, j - 1)])) /
        N;
      p[IX(i, j)] = 0;
    }
  }

  set_bnd(0, div);
  set_bnd(0, p);
  lin_solve(0, p, div, 1, 6);

  for (let j = 1; j < N - 1; j++) {
    for (let i = 1; i < N - 1; i++) {
      velocX[IX(i, j)] -= 0.5 * (p[IX(i + 1, j)] - p[IX(i - 1, j)]) * N;
      velocY[IX(i, j)] -= 0.5 * (p[IX(i, j + 1)] - p[IX(i, j - 1)]) * N;
    }
  }

  set_bnd(1, velocX);
  set_bnd(2, velocY);
}
```

The project function serves to simulate the incompressible fluid, instead of compressible fluid. What I meant by incompressible fluid is something like water(H20), where you cant compress them. The reason why we want to simulate an incompressible fluid is because in the program, we can maintain a constant amount of fluid within the canvas. If it is compressible, the simulated compressible fluid can be too tiny for poeople to see, and which defeats the point of renderign a simulated liquid.

```
function setBoundy(b, x) {
  for (let i = 1; i < N - 1; i++) {
    x[IX(i, 0)] = b == 2 ? -x[IX(i, 1)] : x[IX(i, 1)];
    x[IX(i, N - 1)] = b == 2 ? -x[IX(i, N - 2)] : x[IX(i, N - 2)];
  }

  for (let j = 1; j < N - 1; j++) {
    x[IX(0, j)] = b == 1 ? -x[IX(1, j)] : x[IX(1, j)];
    x[IX(N - 1, j)] = b == 1 ? -x[IX(N - 2, j)] : x[IX(N - 2, j)];
  }

  x[IX(0, 0)] = 0.5 * (x[IX(1, 0)] + x[IX(0, 1)]);
  x[IX(0, N - 1)] = 0.5 * (x[IX(1, N - 1)] + x[IX(0, N - 2)]);
  x[IX(N - 1, 0)] = 0.5 * (x[IX(N - 2, 0)] + x[IX(N - 1, 1)]);
  x[IX(N - 1, N - 1)] = 0.5 * (x[IX(N - 2, N - 1)] + x[IX(N - 1, N - 2)]);
}
```

The setBoundry function is a way to make sure that the fluid itself is not going beyond the size of the canvas itself. The idea is to treat the outer layer of the canvas box as a block that blocks the fluid from going out. So, whenever a fluid hits the canvas box, the outer layer canvas box will counter the fluid with the same amount of velocity, which perfectly simualate the fluid blocker that we wants.

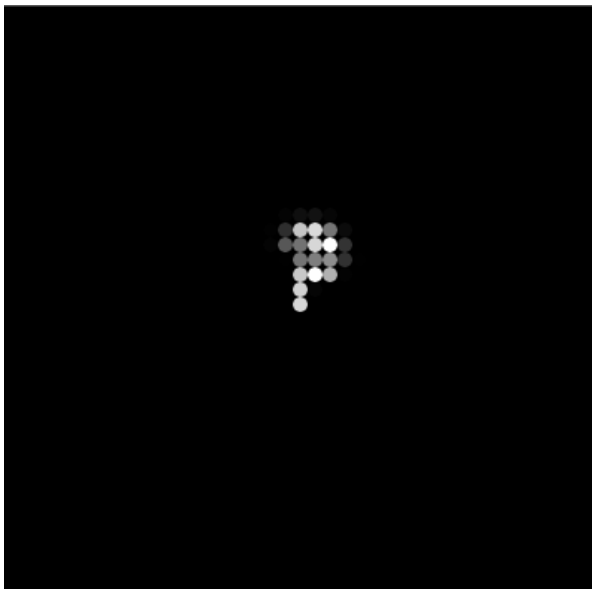
```
function advect(b, d, d0, velocX, velocY, dt) {
  let i0, i1, j0, j1;

  let dtx = dt * (N - 2);
  let dty = dt * (N - 2);

  let s0, s1, t0, t1;
  let tmp1, tmp2, tmp3, x, y;

  let Nfloat = N;
  let ifloat, jfloat;
  let i, j;

  for (j = 1, jfloat = 1; j < N - 1; j++, jfloat++) {
    for (i = 1, ifloat = 1; i < N - 1; i++, ifloat++) {
      tmp1 = dtx * velocX[IX(i, j)];
      tmp2 = dty * velocY[IX(i, j)];
      x = ifloat - tmp1;
      y = jfloat - tmp2;

      if (x < 0.5) x = 0.5;
      if (x > Nfloat + 0.5) x = Nfloat + 0.5;
      i0 = Math.floor(x);
      i1 = i0 + 1.0;
      if (y < 0.5) y = 0.5;
      if (y > Nfloat + 0.5) y = Nfloat + 0.5;
      j0 = Math.floor(y);
      j1 = j0 + 1.0;

      s1 = x - i0;
      s0 = 1.0 - s1;
      t1 = y - j0;
      t0 = 1.0 - t1;

      let i0i = parseInt(i0);
      let i1i = parseInt(i1);
      let j0i = parseInt(j0);
      let j1i = parseInt(j1);

      d[IX(i, j)] =
        s0 * (t0 * d0[IX(i0i, j0i)]) +
        t1 * d0[IX(i0i, j1i)] +
        s1 * (t0 * d0[IX(i1i, j0i)]) +
        t1 * d0[IX(i1i, j1i)];
    }
  }

  set_bnd(b, d);
```

The advert function is responsible to move the fluid around. A good way to explain this function is that it basically binds the variable time with the fluid's variable such as the diffusion, velocity, and density together to enable a simulation of movement within the canvas page.

```
buildFluid() {
  let N = this.size;
  let visc = this.visc;
  let diff = this.diff;
  let dt = this.dt;
  let Vx = this.Vx;
  let Vy = this.Vy;
  let Vz = this.Vz;
  let Vx0 = this.Vx0;
  let Vy0 = this.Vy0;
  let Vz0 = this.Vz0;
  let s = this.s;
  let density = this.density;

  diffuse(1, Vx0, Vx, visc, dt);
  diffuse(2, Vy0, Vy, visc, dt);

  project(Vx0, Vy0, Vx, Vy);

  advect(1, Vx, Vx0, Vx0, Vy0, dt);
  advect(2, Vy, Vy0, Vx0, Vy0, dt);

  project(Vx, Vy, Vx0, Vy0);

  diffuse(0, s, density, diff, dt);
  advect(0, density, s, Vx, Vy, dt);
}
```

The buildFluid function basically is the function that builds the fluid entirely, by using the functions that we have previosly mentioned.

Presentation of my fluid simulation



The fluid is moving accordingly, and it works as intended. I just realized that the GIF for the fluid simulation is not displaying accordingly in the pdf, so I added the gif file on the folder. To read the document with the fluid simulation GIF, you can read the document with Word text editor.

Future works

In the future, I would like to have a 3-dimensional rendering for my fluid simulation project. I did attempt on it, but I did not get very far with in. By the time I have already completed the assignment it is 3 hours before the submission due.

Reflection

I am proud to say that with enough material sources, and lots of time being spent on the assignment, I have successfully managed to implement the fluid effect within a 2d canvas with a bit of a sprinkle for 3d effect without Z-buffer. In my final project, I used the Navier-Stokes's equation, to simulate the fluid simulation. It is important to keep in mind that I tried to reduce the size of the canvas and the frame rate due to frame speed rendering can be very slow. The project takes a tremendous amount of time for me to complete, because I will have to learn how to use the new graphical library framework, while also working on the project.

Installation Guide

To run the program, first open the program file with a VSCode, then download the "P5.vscode". Once you are done opening the file in VSCode, go to your bottom right of the VS code, and click on the "go live" button. Picture as below, and once click you should be redirect to a browser running the program.