UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

COMPUTACION GRAFICA

Simulación de Fluidos

Alumnos:
Chayña Batallanes Josnick
Perez Rodriguez Angelo Aldo
Pucho Zevallos Kelvin Paul
Vilcapaza Flores Luis Felipe
Sihuinta Perez Luis Armando

Julio 2021

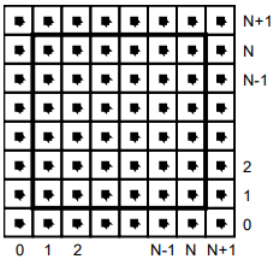
Índice

1. Introducción

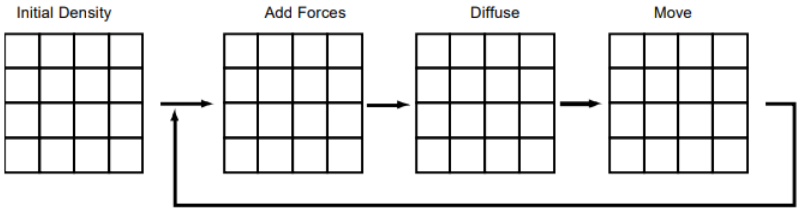
En la Computación Gráfica la simulación de fluidos es algo que cada día va tomando más importancia en este ámbito. Los flujos de fluidos están en todas partes: desde el aumento del humo, las nubes y la niebla hasta el flujo de los ríos y océanos. También debemos tomar en cuenta que estos aparecen en los videojuegos. Cada vez se trata de hacer los fluidos de forma más realista, y para animarlos la tarea que se lleva acabo no es muy fácil.
Para el trabajo siguiente se va a realizar un simulación de humo usando la ecuación de Navier-Stokes.

2. Descripción

Para conseguir una Simulación de Fluidos estable la alternativa útil es la física de los flujos con la ecuación de Navier-Stokes. El cual es un modelo matemático preciso para la mayoría de los flujos de fluidos. Esta ecuación ayuda a simular fluidos de forma visual tal que el usuario puede interactuar con ella de manera real como se produce estos fluidos.



La teoría de la siguiente imagen no solo se queda limitada en el 2D



La estructura principal del solucionador sigue la de la ecuación. Primero comenzamos con una cuadrícula inicial de densidades que generalmente está vacía. Luego actualizamos esta cuadrícula en tres pasos. Cada paso corresponde a uno de los términos de la ecuación. Primero agregamos las fuentes a la cuadrícula. Esto es realmente sencillo. En mi implementación, las fuentes las proporciona una cuadrícula. Entonces, todo lo que tengo que hacer es multiplicar esta cuadrícula por el paso de tiempo y agregarlos a la cuadrícula de densidad. Déjame saber explicarte cómo resolver el paso de difusión.

2.1. ¿Que es la ecuación de Navier-Stoke?

En 1822, el matemático e ingeniero francés Claude-Louis Navier deduce un sistema de ecuaciones que describe el comportamiento de algunos fluidos. Veinte años después, Sir George Gabriel Stokes, partiendo de un modelo diferente, completa la descripción de esas ecuaciones, bautizadas como ecuaciones de Navier-Stokes en honor a ambos. Simplificando, digamos que se obtienen aplicando los principios de conservación de la mecánica y la termodinámica a un volumen fluido. Esto se suele trabajar con ellas a partir de su formulación diferencial, como la que se usara posteriormente. Las ecuaciones de Navier-Stokes expresan matemáticamente la conservación del momento y la conservación de la masa para los fluidos newtonianos. Aplicando los principios de conservación de la mecánica y la termodinámica a un volumen fluido se obtiene la llamada formulación integral de las ecuaciones. Para llegar a su formulación diferencial, se manipulan aplicando ciertas consideraciones, principalmente aquella en la que los esfuerzos tangenciales guardan una relación lineal con el gradiente de velocidad (ley de viscosidad de Newton), obteniendo de esta manera la formulación diferencial que generalmente es más útil para la resolución de los problemas que se plantean en la mecánica de fluidos. Esta formulación de Navier-Stokes como se dijo antes son un conjunto de ecuaciones en derivadas parciales no lineales que describen el movimiento de un fluido viscoso e incompresible.
Así que se reutilizara la abundante literatura en física e ingeniería, ya que una de esas ecuaciones es resolver la ecuación lineal y no lineal que se va hallar por medio de las siguientes ecuaciones de Física de Fluidos:

La ecuaciones de Navier Stokes para la velocidad respecto al tiempo en una notación vectorial compacta:

$$\frac{\partial u}{\partial t} = -(u.\nabla)u + v\nabla^2 u + f$$

(1)

La ecuación para una densidad que se mueve a través del campo de velocidad:

$$\frac{\partial \rho}{\partial t} = -(u.\nabla)\rho + k\nabla^2 \rho + S$$

(2)

Por medio de estas ecuaciones podemos notar que el estado de un fluido en un instante de tiempo determinado se modela como un campo vectorial de velocidad, una función que asigna un vector de velocidad a cada punto del espacio.

Dado el estado actual de la velocidad y un conjunto actual de fuerzas, Estas ecuaciones indican con precisión cómo cambiará la velocidad en un paso de tiempo infinitesimal. La ecuación a grandes rasgos, indica que el cambio de velocidad se debe a los tres términos situados a la derecha del signo de igualdad.
Un campo de velocidad empieza a mover objetos como partículas de humo, polvo , etc. El movimiento de estos objetos se calcula convirtiendo las velocidades que rodean al objeto en

fuerzas del cuerpo. Los objetos ligeros, como el polvo, suelen ser simplemente se ven arrastrados por el campo de velocidad: simplemente siguen la velocidad. La densidad suele tomar valores entre cero y uno: donde no hay humo la densidad es cero, y en otros lugares indica la cantidad de partículas presentes. La evolución del campo de densidad a través del campo de velocidad del fluido también puede ser descrita por una ecuación matemática precisa, que se representa en la ecuacion 2.
La ecuación de la densidad es más sencilla que la de la velocidad. La razón técnica es que la primera es lineal mientras que la segunda es no lineal.
Primero desarrollamos un algoritmo para la densidad que se mueve a través de un campo de velocidad fijo y luego esto se puede aplicar para calcular también la evolución del campo de velocidad. Por lo tanto primero se procederá a resolver la ecuación de la densidad. Donde se encontrara con muchos componentes que reutilizara para resolver la más difícil de la ecuación para la simulación de la velocidad.

2.2. Descripción de la Ecuaciones de Navier-Stokes

Las primeras tecnicas eran limitadas sobre todo a 2D y estas utilizaban técnicas como los vortex blobs,pero en el caso de 3D se usa la técnica de voxels. Para explicar detalladamente como resolver las ecuaciones, se empleara un ejemplo para simulación de fluidos en 2D.

- **Ecuación de la densidad**

$$\frac{\partial \rho}{\partial t} = -(u.\nabla)\rho + k\nabla^2\rho + S \tag{3}$$

donde:
$\rho = densidad$
$u = velocidad$
$k = coeficiente\ kappa$
$S = fuente\ externa$

Primero se empezara por citar el ultimo termino de la ecuación que se considera el mas fácil:

$$\frac{\partial \rho}{\partial t} = -(u.\nabla)\rho + k\nabla^2\rho + S \rightarrow \quad S \tag{4}$$

Representa el aumento de la densidad debido a una fuente externa. Y estas fuentes pueden ser proporcionados por el usuario a través de una interfaz de usuario o por objetos en la simulación.

Luego se procederá a explicar el segundo termino el cual se encargara de representar la difusión del fluido respecto a un tamaño de paso.

$$\frac{\partial \rho}{\partial t} = -(u.\nabla)\rho + k\nabla^2\rho + S \rightarrow \boxed{k\nabla^2\rho}$$

(5)

Entonces en forma visual para representar la difusión seria:

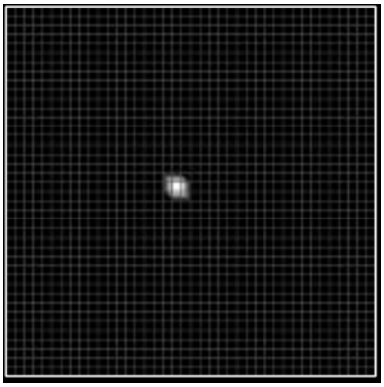


Figura 1: Densidad D^n

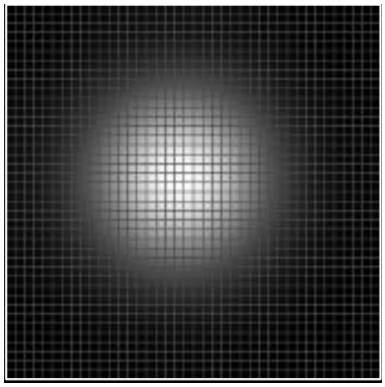


Figura 2: Densidad D^{n+1}

La figura mostrada tiene la idea básica de intercambiar densidades sólo entre los vecinos inmediatos como se muestra el la figura siguiente:

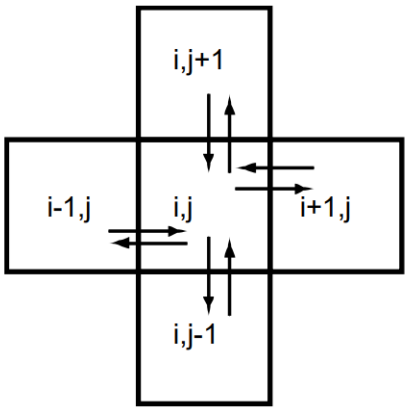


Figura 3: Intercambio de densidades entre vecinos

En cada flujos de todas las caras obtenemos la siguiente regla de actualización. Pero desafortunadamente no funciona.

Entonces quedaría de la siguiente manera:

$$D_{i,j}^{n+1} = D_{i,j}^{n} + k\nabla t(D_{i-1,j}^{n} + D_{i+1,j}^{n} + D_{i,j-1}^{n} + D_{i,j+1}^{n} - 4D_{i,j}^{n})/h^2 \qquad (6)$$

Pero hay una razón por la que no funciona es que puede volverse inestable cuando la tasa de difusión es demasiado alta, el paso de tiempo demasiado grande o el espacio del grid es demasiado pequeño. El problema es que este método colapsa cuando la densidad se propaga más allá de los vecinos.

Cuando es inestable $\nabla tk/h2 > 1/2$ Y para solucionarlo debemos encontrar las densidades que cuando se difunden hacia atrás en el tiempo dan las densidades originales.

Intuitivamente buscamos las densidades que cuando se difunden hacia atrás en el tiempo nos dan las densidades que tenemos actualmente. Por lo tanto nuestra ecuacion 6 se vera distinta la cual es un sistema lineal:

$$D^n_{i,j} = D^{n+1}_{i,j} - k\nabla t(D^{n+1}_{i-1,j} + D^{n+1}_{i+1,j} + D^{n+1}_{i,j-1} + D^{n+1}_{i,j+1} - 4D^{n+1}_{i,j})/h^2 \qquad (7)$$

El problema es que todos los términos del lado izquierdo son desconocidos. Así que terminamos con un sistema lineal que tenemos que resolver, el sistema es disperso y existen muchos solucionadores rápidos. Además el costo para resolver el sistema a lo largo de un gran paso de tiempo es mucho más efectivo que dar muchos pasos pequeños con un solucionador inestable.

Existen Solucionadores lineales que podemos usar:

Nombre	Costo	Comentario
Eliminación Guaseana	N^3	usar solo para N muy pequeño
Relajación Jacobi / SOR	N^2	Fácil de codificar pero lenta
FFT / reducción cíclica	N logN	Úselo cuando no haya límites internos
Gradiente conjugado	N^15	Úselo cuando haya los límites internos
Multi-grid	N	Más lento que FFT en la práctica.
Multi-grid	N	Difícil de codificar cuando existen límites internos

El solucionador más fácil de implementar es la relajación simple como Jacobi o Gauss-Seidel. El único problema es que no convergen tan rápido.

Y finalmente el primer termino el cual a lo largo de un paso de tiempo se quiere mover la densidad a lo largo del campo de velocidad. y este seria:

$$\frac{\partial \rho}{\partial t} = -(u.\nabla)\rho + k\nabla^2\rho + S \rightarrow \boxed{-(u.\nabla)\rho} \qquad (8)$$

De forma visual se muestra en la siguiente figura:

En este caso los flujos estarán sesgados por la dirección del campo de velocidad. La aplicación ingenua de este esquema vuelve a dar lugar a inestabilidades. El problema vuelve a ser cuando la transferencia se produce entre vecinos que están a más de una celda de distancia. Esto ocurre cuando la velocidad es demasiado grande, el paso de tiempo demasiado grande o el espaciado de la malla demasiado pequeño.

Cuando es inestable $\nabla t|u| > h$

La idea básica aquí es demostrar que el movimiento de la densidad se haga parecido al del movimiento de partículas.

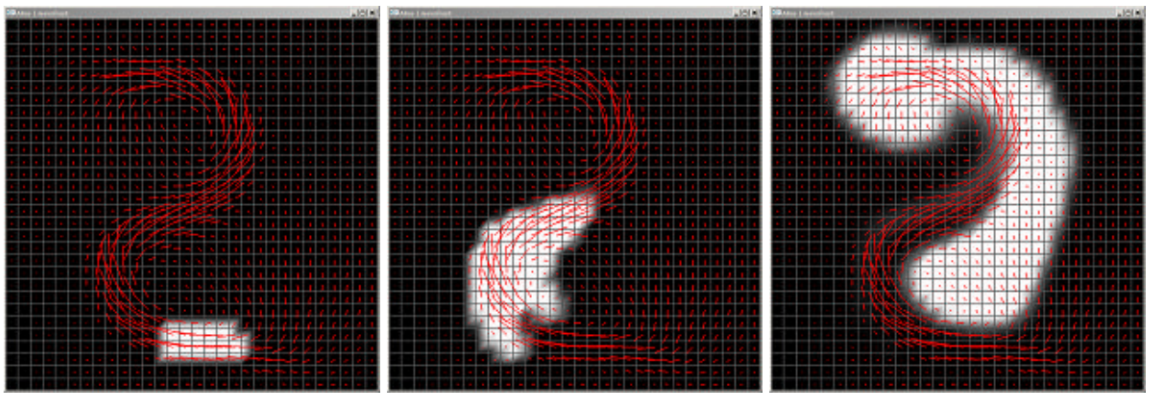


Figura 4: El paso de advección mueve la densidad a través de un campo de velocidad estático

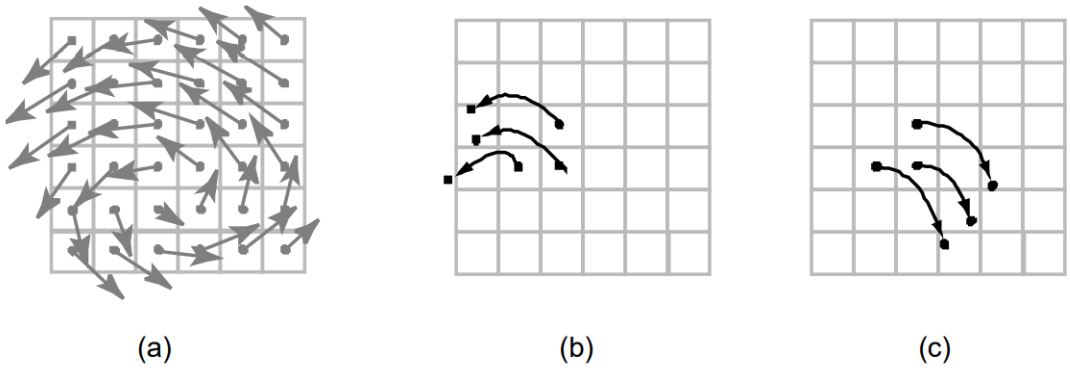


Figura 5: Paso de advección. En lugar de mover los centros de las celdas hacia adelante en tiempo (b) a través del campo de velocidad mostrado en (a), buscamos las partículas que terminan exactamente en los centros de las celdas trazando hacia atrás en el tiempo desde los centros de las celdas (c).

Básicamente es encontrar las posiciones de las partículas que después de un paso de tiempo terminan exactamente en los centros de la cuadrícula. Para encontrar estas partículas, simplemente trazamos cada centro de voxel de la cuadrícula hacia atrás a través del campo. Haciendo esto acabaremos en otro lugar de la cuadrícula. Primero localizamos las cuatro celdas más cercanas al punto. Y luego se interpola la densidad de estos grids. El valor interpolado sera la nueva densidad del grid de salida. Para ello necesitamos dos grids. Una que contenga los valores de densidad del paso de tiempo anterior y otra que contenga los nuevos valores interpolados.
La propiedad importante de esta técnica es que es incondicionalmente estable: no importa lo grande que sea el paso de tiempo, esta técnica no colapsara. Como los nuevos datos son una interpolación lineal de los datos anteriores, tenemos que el nuevo máximo de las nuevas densidades está siempre limitado por la densidad máxima de los valores antiguos. Así que la densidad está siempre limitada sin importar el tamaño del paso de tiempo y por lo tanto nunca explotará y se volverá inestable

- **Ecuación de la velocidad**
  La ecuación de la velocidad también tiene un término de "difusión"junto con la viscosidad y cuando sea más alta, más suave será el campo de velocidad, dando lugar a

fluidos viscosos. En este caso tenemos que resolver 2 ecuaciones de difusión en 2D y tres ecuaciones en 3D, una para cada componente del campo de velocidad. El último término es el término de fuerza y también se puede contabilizar como para las fuentes en el solucionador de la densidad.
Finalmente, el primer término es el más complicado. Es igual que el término correspondiente a la densidad, salvo que la velocidad aparece dos veces, por lo que no es lineal. En el caso de la densidad, este término establece que la densidad debe seguir el campo de la velocidad. Así que podemos interpretar este término como si dijera que la "velocidad debería moverse a lo largo de sí misma".

$$\frac{\partial u}{\partial t} = -(u.\nabla)u + v\nabla^2 u + f \tag{9}$$

donde:
$\rho = densidad$
$u = velocidad$
$v = viscosidad\ kappa$
$f = fuerzas\ externas$

Primero se empezara por citar el ultimo termino de la ecuación el cual simplemente agrega fuerzas a cada grid.

$$\frac{\partial u}{\partial t} = -(u.\nabla)u + v\nabla^2 u + f \rightarrow \boxed{f} \tag{10}$$

Esta fuerza sera proporcionada por el usuario o por un objeto.
Sobre el segundo termino que es la difusión de viscosidad seria:

$$\frac{\partial u}{\partial t} = -(u.\nabla)u + v\nabla^2 u + f \rightarrow \boxed{v\nabla^2 u} \tag{11}$$

Respecto al primer termino que es el mas interesante la cual es:

$$\frac{\partial u}{\partial t} = -(u.\nabla)u + v\nabla^2 u + f \rightarrow \boxed{-(u.\nabla)u} \tag{12}$$

Aquí nuevamente necesitamos dos celdas para la velocidad. Una que contenga los valores antiguos y otra que contenga los nuevos valores interpolados. y Así como para hallar la densidad, trazamos cada punto de la celda hacia atrás en el tiempo utilizando las velocidades antiguas. Una ves estando en otra posición de la celda . Y en cuanto a la densidad interpolamos una nueva velocidad en ese lugar a partir de las celdas del grid vecino.

Y luego establecemos la nueva velocidad a la interpolada. Este método ya se uso mas antes la cual es estable al igual que para el solucionador de densidad.

Dicha técnica usada fue inventada por primera vez en 1952 por Courant, Rees e Isaacson y ha sido redescubierta por muchas investigaciones en diferentes campos. Es más conocida como técnica semi-Lagrangian.

- **Conservación de la masa**
  Finalmente realizado la difusión y la adveccion tanto de la densidad como de la velocidad debemos considerar una propiedad importante de los fluidos donde indica que todo fluido es Ïncompresible". Así que todavía hay un paso que tenemos que cumplir antes de terminar y es que el fluido debe conservar la masa.

  Esto significa Que el flujo que entra en un grid sea igual al que sale de ella. En el algoritmo se llamara a este método después de los tres pasos anteriores,
  Flujo de entrada a la celda = Flujo de salida de la celda

$$\frac{\partial u}{\partial t} = -(u.\nabla)u + v\nabla^2 u + f$$

$$\mathrm{U}_{i+1,j} - U_{i-1,j} + V_{i,j+1} - V_{i,j-1} = 0not \tag{13}$$

Para ello e utiliza un resultado matemático conocido como la descomposición de Hodge de un campo vectorial. Este resultado establece que todo campo vectorial es la suma de un campo que conserva la masa y un campo de gradiente.

Entonces para conseguir un campo que sea mas realista a una simulación de fluidos necesitamos en el campo de conservación de la masa ya que tiene bonitos vórtices que darán lugar a flujos de aspecto arremolinado.

En cambio, el campo de gradiente es el peor caso posible: en cada punto, el flujo se dirige hacia dentro o hacia fuera. El campo de gradiente puede visualizarse como la función de pendiente de algún campo de altura.

Entonces para obtener un campo que conserve la masa a partir de un campo vectorial arbitrario, basta con restarle la parte del gradiente. Como se muestra en la figura siguiente:

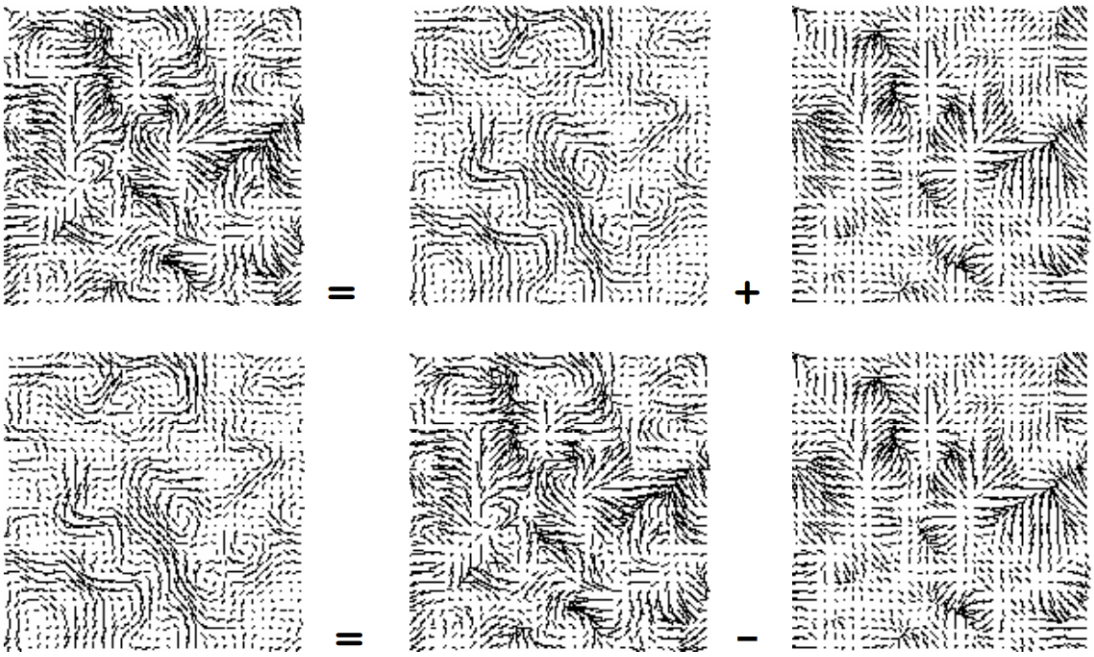


Figura 6: Todo campo de velocidad es la suma de un campo incompresible y un campo de gradiente .

Donde el campo escalar satisface a una ecuación y resulta que ese campo de gradiente se puede calcular resolviendo con una ecuación de Poisson. Y para esto también es un sistema lineal simétrico disperso que podemos resolver usando cualquiera de los solucionadores mencionados en la tabla.

3. Algoritmos de Implementación

Esta se dividirá en partes importantes las cuales describen las ecuaciones tratadas anteriormente

3.1. Método para obtener los indices de cada voxels

```
function IX(x, y, z) {
    return x + y * N + z * N * N;
}
```

3.2. Método para agregar densidad y velocidad

```

FluidCubeAddDensity(x, y,z, amount) {
    this.density[IX(x, y, z)] += amount;
}
```

```
FluidCubeAddVelocity(x,y,z,amountX,amountY,amountZ) {

    let index = IX(x, y, z);
    this.Vx[index] += amountX;
    this.Vy[index] += amountY;
    this.Vz[index] += amountZ;
}
```

3.3. Método para establecer el límite

El propósito de la rutina $set_bnd()$ que aparece en muchos lugares de nuestro código. Es porque suponiendo que el fluido está contenido en una caja con paredes sólidas: ningún flujo debe salir de las paredes. Esto significa simplemente que la componente horizontal de la velocidad debe ser cero en las paredes verticales, mientras que la componente vertical de la velocidad debe ser nula en las paredes horizontales. Para la densidad y otros campos considerados en el código simplemente asumimos la continuidad. A continuación el siguiente código implementado:

```
function set_bnd(b,x)
{
    for(let j = 1; j < N - 1; j++) {
        for(let i = 1; i < N - 1; i++) {
            x[IX(i, j, 0  )] = b == 3 ? -x[IX(i, j, 1  )] : x[IX(i, j, 1
                )];
            x[IX(i, j, N-1)] = b == 3 ? -x[IX(i, j, N-2)] : x[IX(i, j, N
                -2)];
        }
    }
    for(let k = 1; k < N - 1; k++) {
        for(let i = 1; i < N - 1; i++) {
            x[IX(i, 0  , k)] = b == 2 ? -x[IX(i, 1  , k)] : x[IX(i, 1  ,
                k)];
            x[IX(i, N-1, k)] = b == 2 ? -x[IX(i, N-2, k)] : x[IX(i, N-2,
                k)];
        }
    }
    for(let k = 1; k < N - 1; k++) {
        for(let j = 1; j < N - 1; j++) {
            x[IX(0  , j, k)] = b == 1 ? -x[IX(1  , j, k)] : x[IX(1  , j,
                k)];
            x[IX(N-1, j, k)] = b == 1 ? -x[IX(N-2, j, k)] : x[IX(N-2, j,
                k)];
        }
    }

    x[IX(0, 0, 0)]       = 0.33 * (x[IX(1, 0, 0)]
                                 + x[IX(0, 1, 0)]
                                 + x[IX(0, 0, 1)]);
    x[IX(0, N-1, 0)]     = 0.33 * (x[IX(1, N-1, 0)]
                                 + x[IX(0, N-2, 0)]
                                 + x[IX(0, N-1, 1)]);
    x[IX(0, 0, N-1)]     = 0.33 * (x[IX(1, 0, N-1)]
```

```
                                    + x[IX(0, 1, N-1)]
                                    + x[IX(0, 0, N)]);
    x[IX(0, N-1, N-1)]   = 0.33 * (x[IX(1, N-1, N-1)]
                                    + x[IX(0, N-2, N-1)]
                                    + x[IX(0, N-1, N-2)]);
    x[IX(N-1, 0, 0)]     = 0.33 * (x[IX(N-2, 0, 0)]
                                    + x[IX(N-1, 1, 0)]
                                    + x[IX(N-1, 0, 1)]);
    x[IX(N-1, N-1, 0)]   = 0.33 * (x[IX(N-2, N-1, 0)]
                                    + x[IX(N-1, N-2, 0)]
                                    + x[IX(N-1, N-1, 1)]);
    x[IX(N-1, 0, N-1)]   = 0.33 * (x[IX(N-2, 0, N-1)]
                                    + x[IX(N-1, 1, N-1)]
                                    + x[IX(N-1, 0, N-2)]);
    x[IX(N-1, N-1, N-1)] = 0.33 * (x[IX(N-2, N-1, N-1)]
                                    + x[IX(N-1, N-2, N-1)]
                                    + x[IX(N-1, N-1, N-2)]);
}
```

3.4. Metodo para resolver la ecuacion lineal

Este metodo sera llamado tanto para hallar la densidad original como para la velocidad original.

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

3.5. Método para la Difusión

Este método también sera utilizado para la difusión de la densidad junto con una coeficiente kappa mientras que la difusión de la velocidad junto con la viscosidad

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

3.6. Método para la Conservación de la Masa

```
function project(velocX,velocY,velocZ,p,div)
{
    for (let k = 1; k < N - 1; k++) {
        for (let j = 1; j < N - 1; j++) {
            for (let i = 1; i < N - 1; i++) {
                div[IX(i, j, k)] = -0.5*(
                         velocX[IX(i+1, j  , k  )]
                        -velocX[IX(i-1, j  , k  )]
                        +velocY[IX(i  , j+1, k  )]
                        -velocY[IX(i  , j-1, k  )]
                        +velocZ[IX(i  , j  , k+1)]
                        -velocZ[IX(i  , j  , k-1)]
                    )/N;
                p[IX(i, j, k)] = 0;
            }
        }
    }
    set_bnd(0, div, N);
    set_bnd(0, p, N);
    lin_solve(0, p, div, 1, 6, iter, N);

    for (let k = 1; k < N - 1; k++) {
        for (let j = 1; j < N - 1; j++) {
            for (let i = 1; i < N - 1; i++) {
                velocX[IX(i, j, k)] -= 0.5 * (  p[IX(i+1, j, k)]
                                                -p[IX(i-1, j, k)]) * N;
                velocY[IX(i, j, k)] -= 0.5 * (  p[IX(i, j+1, k)]
                                                -p[IX(i, j-1, k)]) * N;
                velocZ[IX(i, j, k)] -= 0.5 * (  p[IX(i, j, k+1)]
                                                -p[IX(i, j, k-1)]) * N;
            }
        }
    }
    set_bnd(1, velocX, N);
    set_bnd(2, velocY, N);
    set_bnd(3, velocZ, N);
}
```

3.7. Método para la Advección

Este método también sera citado por la ecuación de la densidad que traslada la masa en función al campo de velocidad, mientras que la ecuación de velocidad tratada en función a si misma.

```
function advect(b,d,d0,velocX, velocY,velocZ,dt)
{
    let i0, i1, j0, j1, k0, k1;

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

    let s0, s1, t0, t1, u0, u1;
    let tmp1, tmp2, tmp3, x, y, z;

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

    for(k = 1, kfloat = 1; k < N - 1; k++, kfloat++) {
        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, k)];
                tmp2 = dty * velocY[IX(i, j, k)];
                tmp3 = dtz * velocZ[IX(i, j, k)];
                x    = ifloat - tmp1;
                y    = jfloat - tmp2;
                z    = kfloat - tmp3;

                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;
                if(z < 0.5) z = 0.5;
                if(z > Nfloat + 0.5) z = Nfloat + 0.5;
                k0 = Math.floor(z);
                k1 = k0 + 1.0;

                s1 = x - i0;
                s0 = 1.0 - s1;
                t1 = y - j0;
                t0 = 1.0 - t1;
                u1 = z - k0;
                u0 = 1.0 - u1;

                let i0i = parseInt(i0);
                let i1i = parseInt(i1);
                let j0i = parseInt(j0);
                let j1i = parseInt(j1);
                let k0i = parseInt(k0);
                let k1i = parseInt(k1);

                d[IX(i, j, k)] =

```

```
                    s0 * ( t0 * (u0 * d0[IX(i0i, j0i, k0i)]
                                +u1 * d0[IX(i0i, j0i, k1i)])
                         +( t1 * (u0 * d0[IX(i0i, j1i, k0i)]
                                +u1 * d0[IX(i0i, j1i, k1i)])))
                +s1 * ( t0 * (u0 * d0[IX(i1i, j0i, k0i)]
                                +u1 * d0[IX(i1i, j0i, k1i)])
                         +( t1 * (u0 * d0[IX(i1i, j1i, k0i)]
                                +u1 * d0[IX(i1i, j1i, k1i)])));
            }
        }
    }
    set_bnd(b, d, N);
}
```

3.8. Algoritmo de implementacion en Three.js

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
        >
    <title>Fluid Simulation</title>
</head>
<body>

    <script type = 'module'>
         import * as THREE from './jsm/three.module.js';
        import { GUI } from './jsm/dat.gui.module.js';
        import {OrbitControls} from './jsm/OrbitControls.js';
        import { FluidCube,IX } from './Fluids3D/
           Navier_Stokes_Equations_2.js';
        var N = 20;
        var fluid = new FluidCube(0,0.0000001,0.2);//diffusion,viscosity,
           dt
        function getRndInteger(min,max) {
            return Math.random() * (max - min) + min;
        }

        //camera
        var camera = new THREE.PerspectiveCamera(
            75, // angulo
            window.innerWidth/window.innerHeight, // aspect, es lo que ve
                la camara
            0.1, // near
            2000 // far
        );
        camera.position.z = 50;

        //scene
        var scene = new THREE.Scene();
        scene.background = new THREE.Color(0x008800);

```

```
35
36          //render
37          var renderer = new THREE.WebGLRenderer();
38          renderer.setSize( window.innerWidth, window.innerHeight );
39          document.body.appendChild( renderer.domElement );
40
41          const axesHelper = new THREE.AxesHelper( 15 );
42          scene.add( axesHelper );
43
44          //renderer.render( scene, camera );
45          renderer.render( scene, camera );
46
47          // para los controles del mouse
48          var controls = new OrbitControls( camera, renderer.domElement );
49          controls.minDistance = 3; // minima distancia a q puede hacer
                zoom
50          controls.maxDistance = 100; // maxima distancia a q puede hacer
                zoom
51
52          // para que el renderer se actualize al redimensionar el
                navegador
53          window.addEventListener('resize', redimensionar);
54          function redimensionar(){
55              // actualizamos las vcariables que dependen del tamanio[U+FFFD]
                    navegador
56              camera.aspect = window.innerWidth/window.innerHeight;
57              camera.updateProjectionMatrix();
58              renderer.setSize( window.innerWidth, window.innerHeight );
59              renderer.render( scene, camera );
60          }
61
62          let source = 10.0;
63          let force = 200.0;
64          var animate = function(){
65              requestAnimationFrame(animate);
66
67              for (let i_ = -1; i_ <= 1; i_++) {
68                  for (let j_ = -1; j_ <= 1; j_++) {
69                      for (let k_ = -1; k_ <= 1; k_++) {
70                          fluid.FluidCubeAddDensity((N/2)+i_,(N/2)+j_,(N/2)
                                +k_, getRndInteger(50,150));
71                      }
72                  }
73              }
74
75              for (let in_ = 0; in_ < 2 ; in_++) {
76                  var x_ = getRndInteger(-1.0,1.0);
77                  var y_ = getRndInteger(-1.0,1.0);
78                  var z_ = getRndInteger(-1.0,1.0);
79                  fluid.FluidCubeAddVelocity(N/2, N/2,N/2, x_,y_ ,z_);
80              }
81              fluid.FluidCubeStep();
82
83              for (let i = 0; i < N; i++) {
84                  for (let j = 0; j < N; j++) {
85                      for (let k = 0; k < N; k++) {
```

```
86
87                          //color
88                          var density = fluid.density[IX(i,j,k)];
89
90                          if(density >= 1 && density <= 20){
91                              //mesh
92                              const color2 = new THREE.Color( 0x808080 );
93                              //console.log(density)
94                              color2.addScalar(density);
95                              var geometry = new THREE.BoxGeometry
                                    (1.0,1.0,1.0);
96                              var material = new THREE.MeshBasicMaterial( {
                                    color:0x808080,transparent:true,opacity:
                                    color2.getHex()} );
97                              var cube = new THREE.Mesh( geometry, material
                                     );
98
99                              cube.position.x = i
100                             cube.position.y = j
101                             cube.position.z = k
102                             scene.add(cube);
103                         }
104
105                     }
106
107                 }
108
109             }
110             renderer.render( scene, camera );
111         }
112
113         animate();
114     </script>
115
116 </body>
117 </html>
```

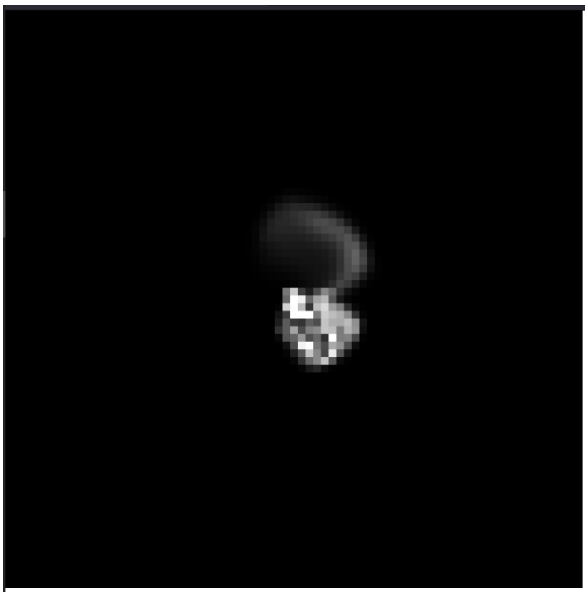
4. Detalles de la Implementación

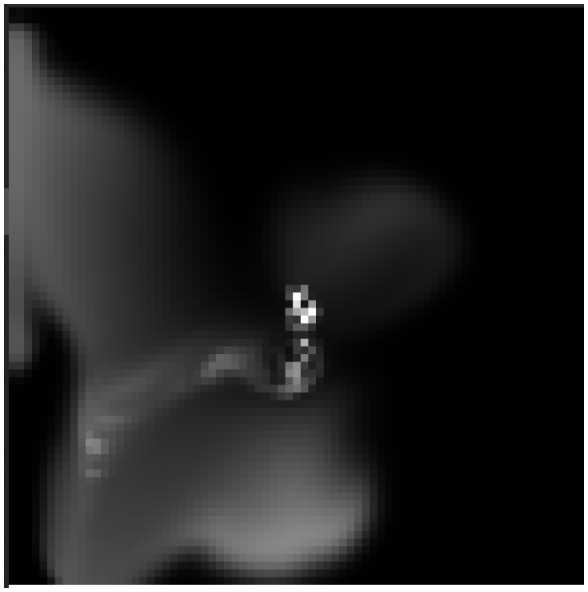
- El algoritmo esta hecho en el lenguaje javascript
- Se utilizo la libreria grafica Three.js
- Enlace github :

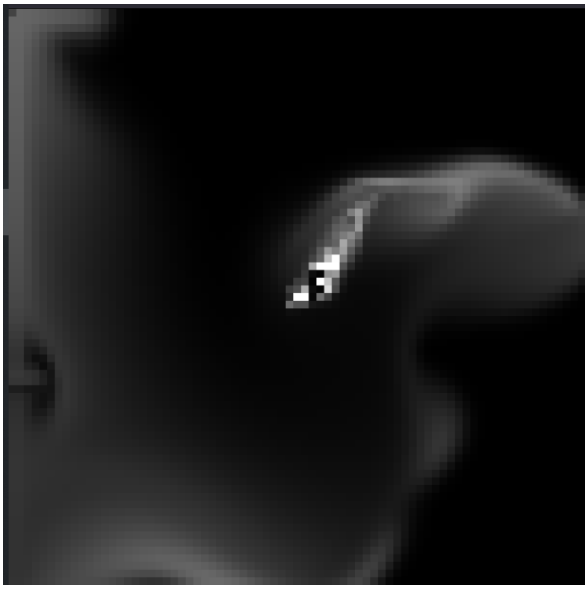
  https://github.com/kpzaolod6000/Graphics-Computing/tree/main/parcial_2/examen

5. Resultados

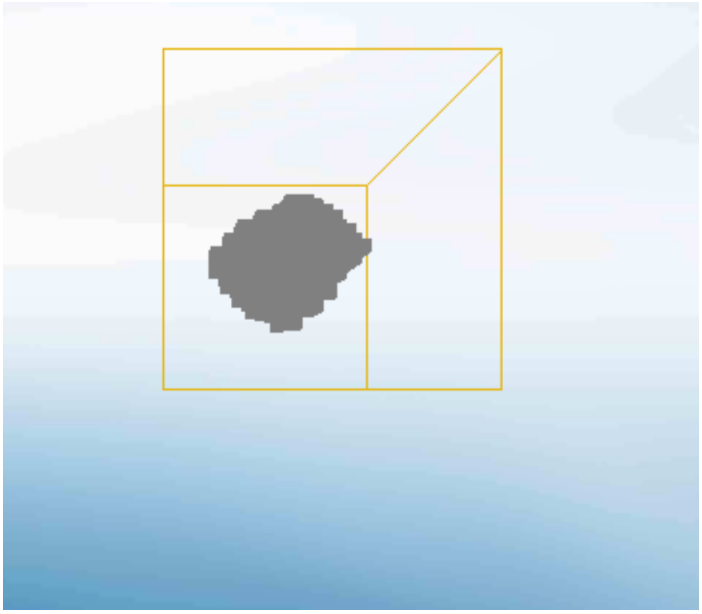
- Capturas de simulación 2D

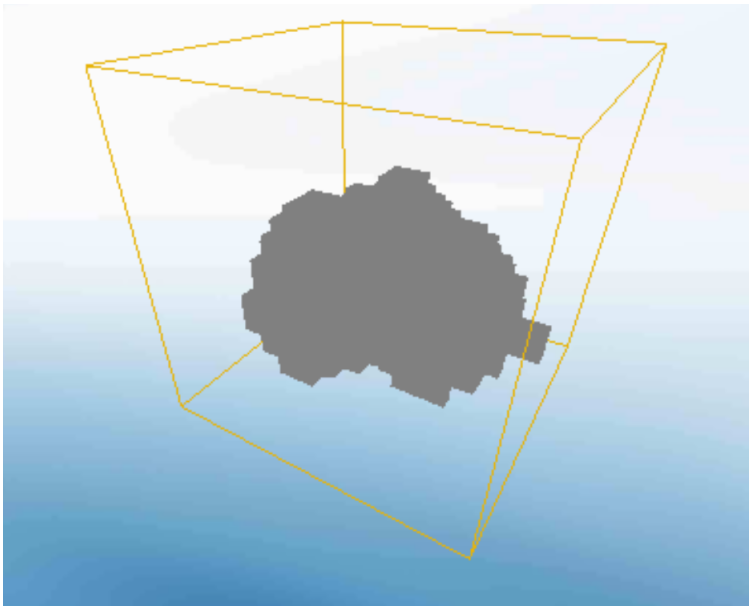


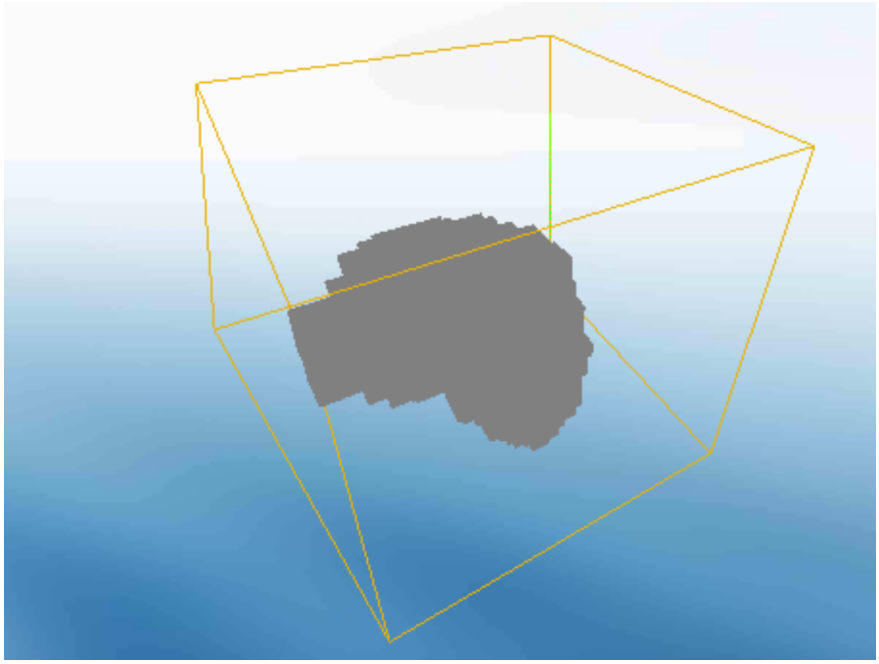




- Capturas de simulación 3D







6. Conclusiones

Existen diferentes aspectos de la forma como se miran los fluidos, y en este pequeño ejemplo implementamos código propuesto de importantes investigadores que demostraron visualmente como se vería una simulación de estos fluidos en 2D y 3D. La fase de investigación sobre las matemáticas que están detrás de estos algoritmos fue bastante riguroso pero se consiguió adquirir un conocimiento base para entender e implementar.