



Tecnológico
de Monterrey

Graficación de Corrientes de Viento

Proyecto Final

Análisis y Diseño de Algoritmos Avanzados

Docente: Dr. José María Aguilera Méndez

Miguel Ángel Pérez Ávila A01369908

Máximo Moisés Zamudio
Chávez A01772056

Andrea Bahena Valdés A01369019

Fecha de entrega: 25 de noviembre del 2024

1. Introducción

Descripción General del Proyecto

Este proyecto genera una graficación de corrientes de viento utilizando el software *Processing*. Dicha graficación se genera a partir de la lectura de datos recopilados en un archivo JSON, y por medio de interpolación bilineal, simula el movimiento de las partículas que conforman las corrientes de viento.

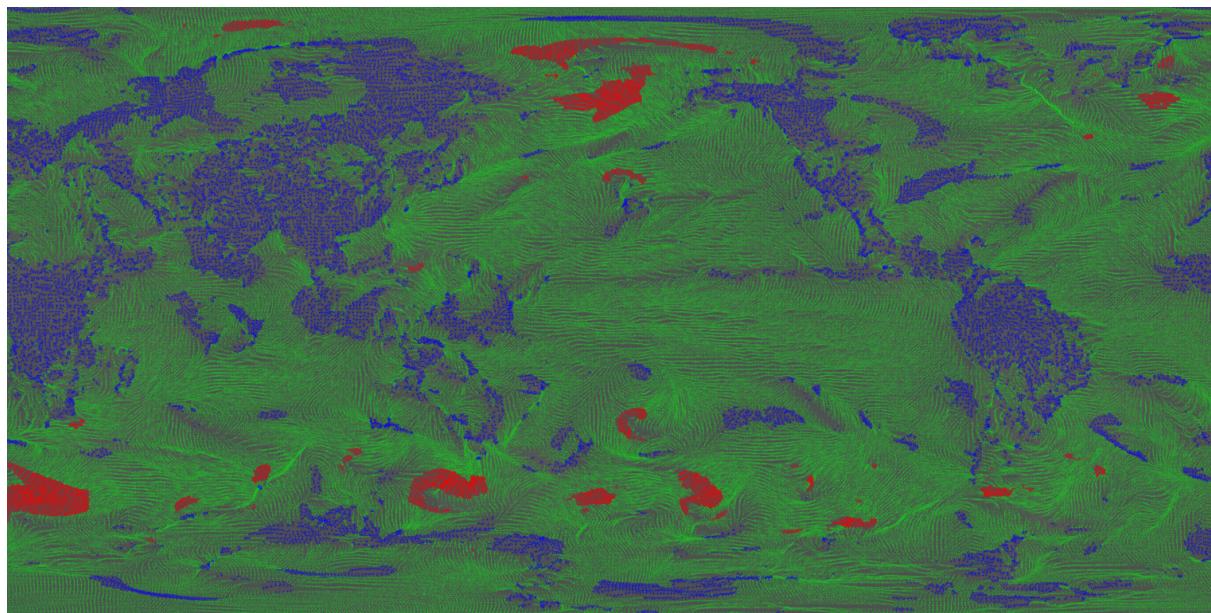


Figura 1 - Simulación de las corrientes

Objetivo Principal

Este proyecto tiene como objetivo principal brindar un acercamiento visual y plenamente intuitivo a la interpretación de datos meteorológicos. Sin embargo, en un futuro también podría ser utilizado como una herramienta de prevención que requeriría poco o nulo trasfondo técnico para su uso.



Figura 2 - Escala de colores

Funcionalidades Principales

- Graficación de las corrientes de viento mediante un sistema de partículas

- Interpolación bilineal para simular el movimiento de las partículas
- Asignación de colores según la magnitud del viento
- Representación sobre el mapa
- Visualización por regiones específicas (zoom y desplazamiento del mapa)

Estructura del Documento

En este documento se explican de forma breve y concisa las tecnologías utilizadas como base del desarrollo. Por otra parte, se explican los principios físicos y matemáticos para la generación de la simulación, así como el proceso de implementación. Los resultados, conclusiones y futuras mejoras también fueron incluidos al final del documento junto con una serie de apéndices útiles.

2. Fundamentos y Base del Desarrollo

Tecnologías Utilizadas

Processing

Se escogió Processing como el software principal para el desarrollo de la simulación, debido a que promueve la alfabetización en software a través de las artes visuales. Es por ello que resulta ser una excelente herramienta para comenzar en el mundo de la programación gráfica. Asimismo, Processing está montado sobre Java, un lenguaje de programación orientado a objetos bastante robusto. Cabe mencionar que existen otras versiones de Processing, como el p5, que está montado sobre JavaScript.



Figura 3 - Otras versiones de Processing

Revisar el Apéndice 1

JSON

El formato JSON (JavaScript Object Notation) es una estructura de datos que organiza la información en dos estructuras clave: **objetos** y **arreglos***. Los formatos JSON

soportan distintos tipos de datos, como cadenas de texto, números, valores booleanos, arreglos, etc.

Objetos - colecciones de pares clave:valor (delimitados por llaves). En otros lenguajes de programación también son llamados diccionarios o tablas de hash.

Arreglos - listas ordenadas de valores (delimitados por corchetes)

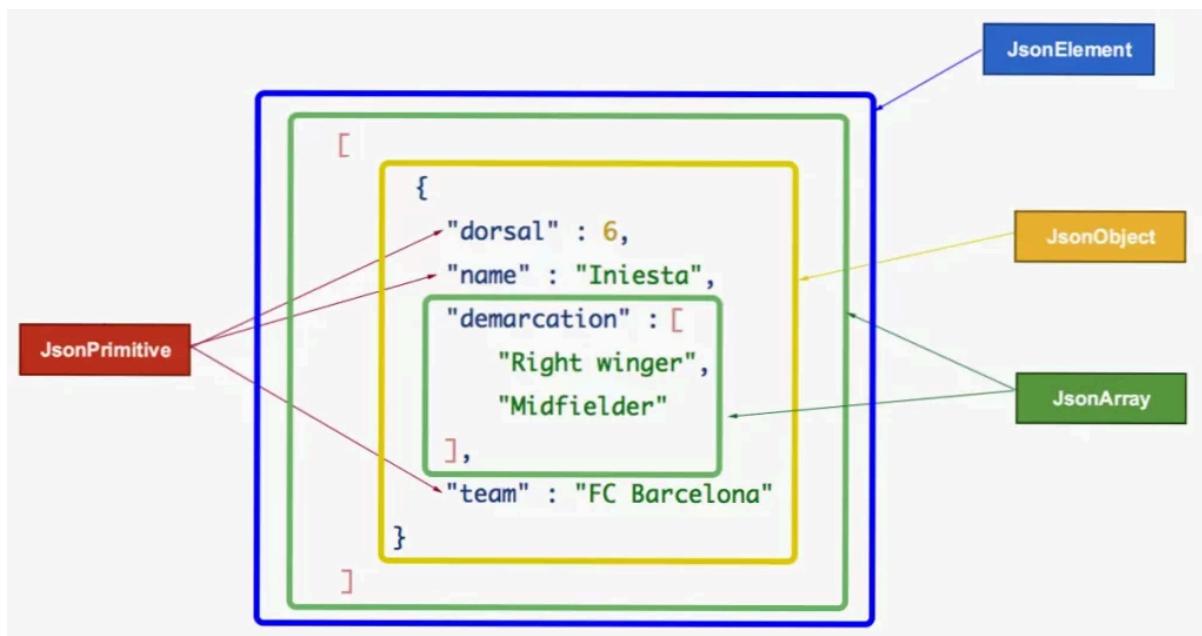


Figura 4 - Estructura de un archivo JSON

Para este proyecto, en el formato JSON se encuentran guardados los componentes vectoriales u y v del viento, así como otro tipo de datos que permiten la generación e interpretación de las partículas que se colocan en el grid.

Otras Bibliotecas

Estas bibliotecas pueden ser instaladas desde el propio entorno de desarrollo de Processing, sin embargo, en caso de tener algún contratiempo, también pueden descargarse de internet e instalarlo en el folder de las librerías de Processing (en la misma ruta de instalación).

Processing Data: Para el procesamiento de archivos JSON

Requisitos Iniciales

Requisitos del Sistema

Procesador: 2.0 GHz o superior

Memoria RAM: 4GB mínimo

Tarjeta gráfica compatible con Processing

Espacio en disco: 500MB mínimo

Requisitos de Software

Processing 4.0 o superior

Java Runtime Environment 8 o superior

Biblioteca JSONObject/JSONArray

Algoritmos y Métodos Clave

Interpolación Bilineal

La interpolación bilineal es un método para estimar valores intermedios utilizando los cuatro puntos más cercanos de una malla. Para imágenes, estos cuatro puntos pueden ser píxeles y lo que se generaría sería un degradado en los colores.

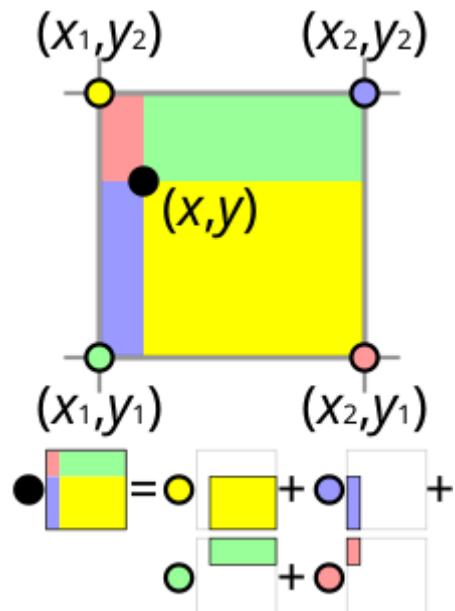


Figura 5 - Representación de la interpolación bilineal

Sin embargo, para efectos de este caso práctico estos cuatro puntos representarán partículas, que generarán la simulación del movimiento de las partículas con los puntos generados a partir del archivo JSON.

La interpolación bilineal en este proyecto se calcula de la siguiente manera:

1. Identifica los cuatro puntos más cercanos del grid
2. Calcula los pesos basados en la distancia
3. Interpola los valores vectoriales
4. Aplica el resultado al movimiento de las partículas

Cálculo de Movimiento de Partículas

El movimiento de las partículas se modela a través de las ecuaciones básicas de movimiento:

$$\begin{aligned}v &= v + a, \\p &= p + v\end{aligned}$$

Figura 6 - Ecuaciones básicas de movimiento

La velocidad por su parte, se actualiza sumando la aceleración y la posición se actualiza sumando la velocidad. Ya que en este sistema físico modelado no se consideran fuerzas que se opongan al movimiento de las partículas, simplemente se asignaron una velocidad y aceleración máxima para el modelado, así como un tiempo de vida para la partícula. En la simulación los valores de la velocidad y aceleración se reinician cuando termina el ciclo de vida de la partícula.

En el sistema también se incorporó un manejo especial de los límites del mapa. Cuando una partícula alcanza los bordes del mapa, reaparece del lado opuesto para continuar con la interpolación.

Manejo de Archivos JSON

Estructura del Archivo JSON

```
[  
  {  
    "header": {  
      ...  
    },  
    "body": {  
      ...  
    },  
    "tail": {  
      ...  
    }  
  }  
]
```

```

    "parameterNumberName": "U-component_of_wind",
    "numberPoints": 65160,
    "nx": 360,                                // Columnas y filas del grid
    "ny": 181,
    ...
},
"data": [                                     // Componente u del viento
    -5.9910207,
    ...
],
{
    "header": {
        ...
        "parameterNumberName": "V-component_of_wind",
        "numberPoints": 65160,
        "nx": 360,                                // Columnas y filas del grid
        "ny": 181,
    },
    "data": [                                     // Componente v del viento
        -1.0850537,
        ...
    ]
}
]

```

Figura 7 - Estructura del Archivo JSON

Como se puede observar, este archivo JSON cuenta con cuatro elementos destacados del header:

- *parameterNumberName*: parte del componente (u,v)
- *numberPoints*: total de puntos recopilados en *data*
- *nx, ny*: dimensiones del grid (transformación de datos unidimensionales a bidimensionales)
- *data*: datos recopilados

Lectura y Procesamiento de Datos en Processing

Los datos se procesan elementalmente mediante cuatro puntos principales:

1. Carga del archivo JSON

2. Parsing de componentes U y V
3. Construcción del campo vectorial
4. Actualización del sistema de partículas

En lo porvenir, se describe el proceso de evolución del código con los cambios más significativos hasta alcanzar los resultados y mejoras esperados,

3. Evolución del Código

Si desea visualizar el historial de versiones, puede hacerlo en el repositorio de GitHub, en la rama *Historial de Versiones*. A continuación se presentan las características relevantes de cada una de las versiones, y cómo contribuyeron para la versión final.

1. Test1 - Lectura y graficación inicial

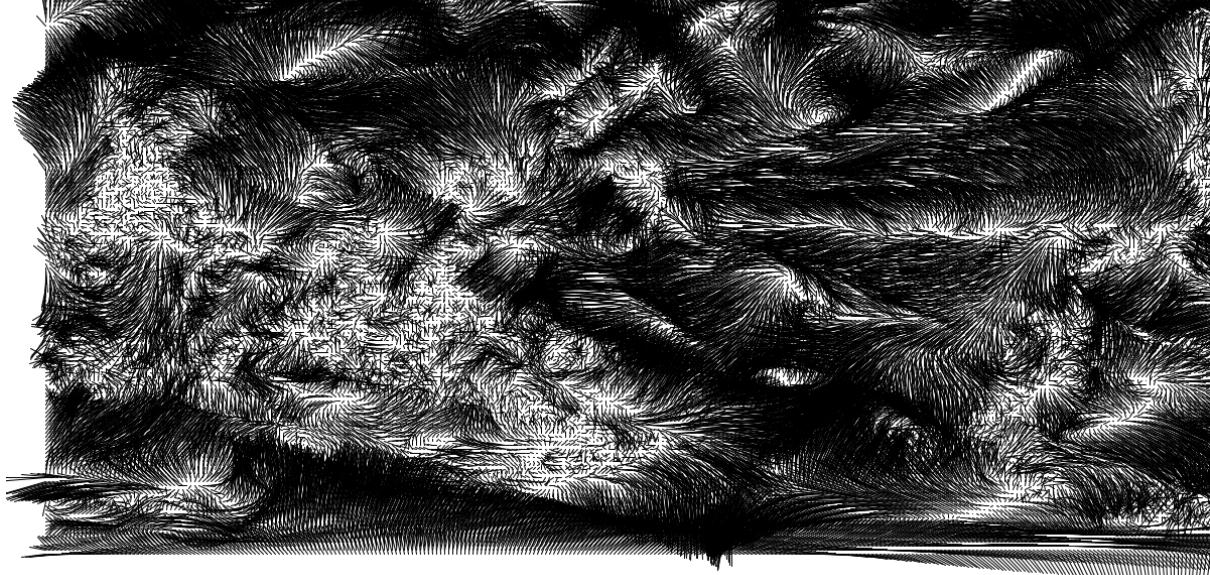


Figura 8 - Simulación de Test1

FUNCIONES PRINCIPALES

- a. Lectura del archivo JSON para obtener los componentes U y V
- b. Parseo de datos para obtener las magnitudes de los vectores
- c. Simulación de movimiento con la interpolación
- d. Se generó una matriz para los componentes u, v (sustituida con PVector)

Parte relevante del código:

```
// Lectura del JSON
String jsonString =
join(loadStrings("current-wind-surface-level-gfs-1.0.json"), "");
JSONArray json = JSONArray.parse(jsonString);

// Leer componentes U y V
JSONObject uComponent = json.getJSONObject(0);
JSONObject vComponent = json.getJSONObject(1);

dataU = uComponent.getJSONArray("data");
dataV = vComponent.getJSONArray("data");
. . .

// Parseo de datos
for (int y = 0; y < gridRows; y++) {
    for (int x = 0; x < gridCols; x++) {
        int idx = x + y * gridCols;
        if (idx < dataU.size() && idx < dataV.size()) {
            float u = dataU.getFloat(idx);
            float v = dataV.getFloat(idx);
            vectorField[y][x] = new PVector(u, v);
        }
    }
. . .

// Interpolación
PVector interpolateBilinear(int x, int y) {
    // Vectores de las esquinas
    PVector v00 = vectorField[x][y];
    PVector v10 = vectorField[x + 1][y];
    PVector v01 = vectorField[x][y + 1];
    PVector v11 = vectorField[x + 1][y + 1];
    // Factores de interpolación basados en frameCount
    float tx = (frameCount % gridCols) / float(gridCols);
    float ty = (frameCount % gridRows) / float(gridRows);
    // Interpolación bilineal
    PVector a = PVector.lerp(v00, v10, tx);
    PVector b = PVector.lerp(v01, v11, tx);
    return PVector.lerp(a, b, ty);
}
```

La interpolación bilineal se utiliza para suavizar la representación del campo vectorial, calculando valores intermedios entre puntos de datos discretos. Este proceso resulta esencial porque el viento es un fenómeno continuo, y la interpolación permite simular este comportamiento fluido a partir de una cuadrícula de datos limitada.

Test2 - Mapa y color de los vectores

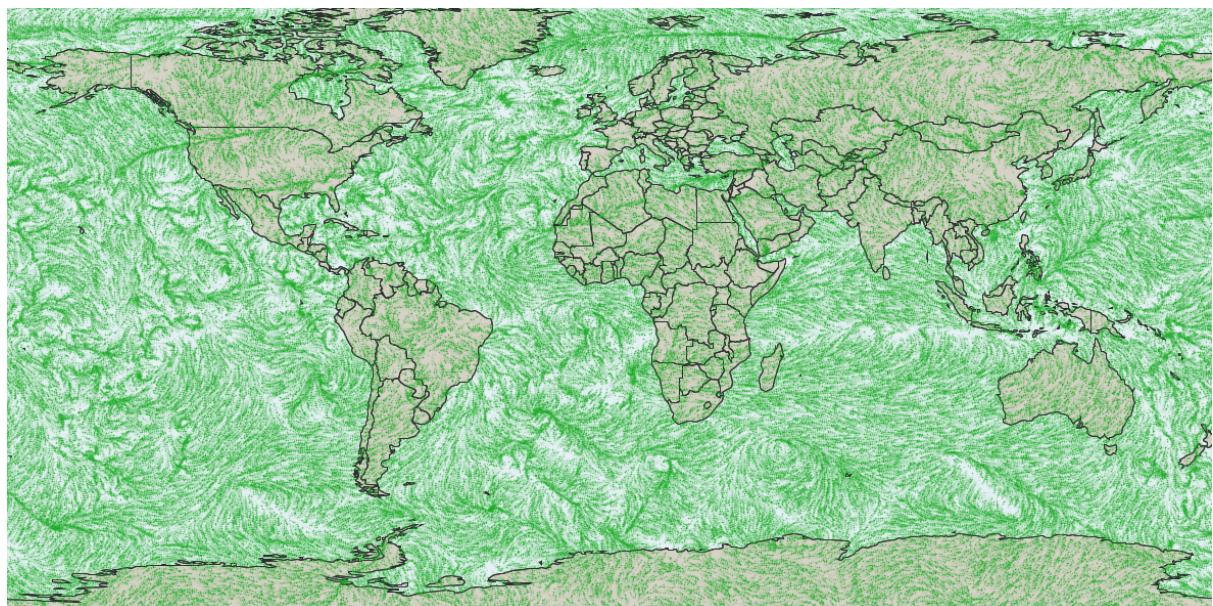


Figura 9 - Simulación de Test2

C A M B I O S P R I N C I P A L E S

- Colocar de fondo un mapa utilizando la librería de **GeoMap**
- Aplicar un color verde para la graficación de los vectores
- Sustituir la matriz de **Test1** por el uso de PiVector para optimizar el manejo de los vectores

```
// Importar y declarar
import org.gicentre.geomap.*;
GeoMap geoMap;
Graphics mapLayer;
// En setup()
GeoMap = new GeoMap(this);
GeoMap.readFile("world");
MapLayer = createGraphics(width, height);
```

```

// En draw()
MapLayer.beginDraw();
MapLayer.background(202, 226, 245, 20);
//Color de las partículas
GeoMap.draw();
MapLayer.endDraw();
Image(mapLayer, 0, 0);
// Color verde para partículas
ParticleColor = new float[]{30, 174, 39, alpha}; // RGB + alpha

```

La implementación del mapa utiliza la librería GeoMap para proporcionar un contexto geográfico a la visualización del viento, mientras que el sistema de capas (PGraphics) permite superponer las partículas verdes que representan las corrientes de viento. Esta combinación crea una visualización más completa del movimiento de las partículas, puesto que supone un marco de referencia de geolocalización.

2. Test3 - Escala de colores para la interpolación

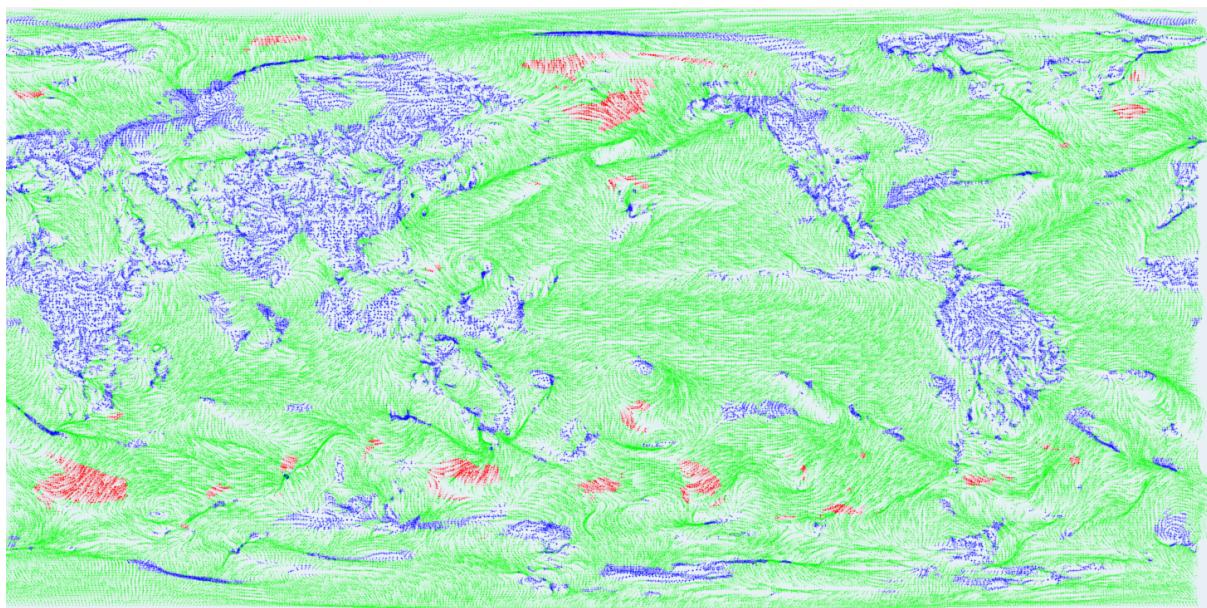


Figura 10 - Simulación de Test3

C A M B I O S P R I N C I P A L E S

- Aplicar la escala de colores basada en la magnitud de las corrientes
- Funciones de zoom y desplazamiento sobre la imagen
- Eliminación del mapa generado con GeoMap

```

// Color basado en magnitud del viento
floatnormalizedMag = map(force.mag(), minMag, maxMag, 0.0, 1.0);
if (normalizedMag > 0.55) { // Rojo - viento fuerte
    particleColor = new float[]{255, 0, 0, alpha};
}
else if (normalizedMag > 0.10) { // Verde - viento medio
    particleColor = new float[]{0, 255, 0, alpha};
}
else { // Azul - viento suave
    particleColor = new float[]{0, 0, 255, alpha};
}

// Control de zoom y desplazamiento
void mouseWheel(MouseEvent event) {
    zoom -= event.getCount() * 0.05;
    zoom = constrain(zoom, minZoom, maxZoom);
}

void mouseDragged() {
    offX += (mouseX - pmouseX) / zoom;
    offY += (mouseY - pmouseY) / zoom;
}

```

Tras implementar esta mejora, se evidenció que la ubicación de los continentes generados mediante los datos del JSON difería del mapa generado por Geomap. Sin embargo, debido a la limitación de Geomap para realizar ajustes en la ubicación, se decidió descartar su uso en el proyecto.

3. Test4 - Capa de transparencia y mapa en imagen

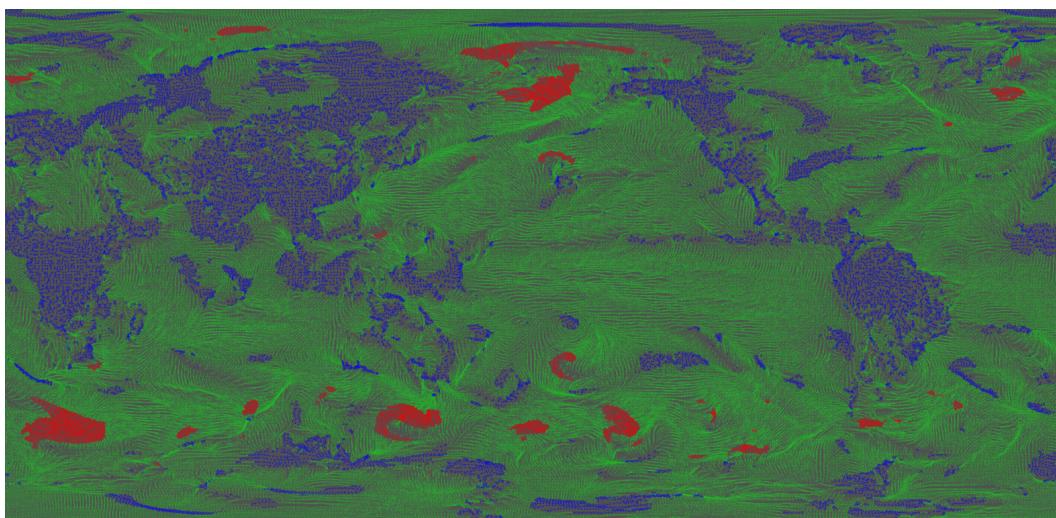


Figura 11 - Simulación de Test4

C A M B I O S P R I N C I P A L E S

- a. Aplicar una capa de transparencia a las partículas
- b. Añadir un mapa de fondo por medio de una imagen
 - i. Problema de renderización
- c. Preservación de la función de zoom
 - i. Errores en el desplazamiento (manejo de capas)

```
// Cargar imagen de fondo
PImage earth;
earth = loadImage("test1.jpg");
earth.resize(width, height);

// En draw()
mapLayer.beginDraw();
mapLayer.noStroke();
mapLayer.fill(80, 25); // Transparencia general
mapLayer.rect(0, 0, width, height);

// Renderizado por capas
translate(offX, offY);
scale(zoom);
translate(-width/2, -height/2);

// Dibujar capas en orden
PImage(earth, 0, 0);
PImage(mapLayer, 0, 0);

// En Particle.show()
void show(PGraphics pg) {
    pg.stroke(particleColor[0], particleColor[1], particleColor[2],
particleColor[3]);
    pg.strokeWeight(1.8);
    pg.point(pos.x, pos.y);
}
```

El manejo de capas y transparencias permite mostrar el viento sobre el mapa de forma clara, sin embargo, supone un problema para el desplazamiento una vez hecho el zoom. Se debe a la traslación de las matrices. La transparencia de las partículas, basada en la magnitud del viento, junto con el zoom y desplazamiento, facilita la

visualización de los patrones de viento en diferentes regiones, y permite que sea visualmente mucho más atractivo y claro el flujo de las corrientes.

4. Test5 - Degradado de colores y mapa estático

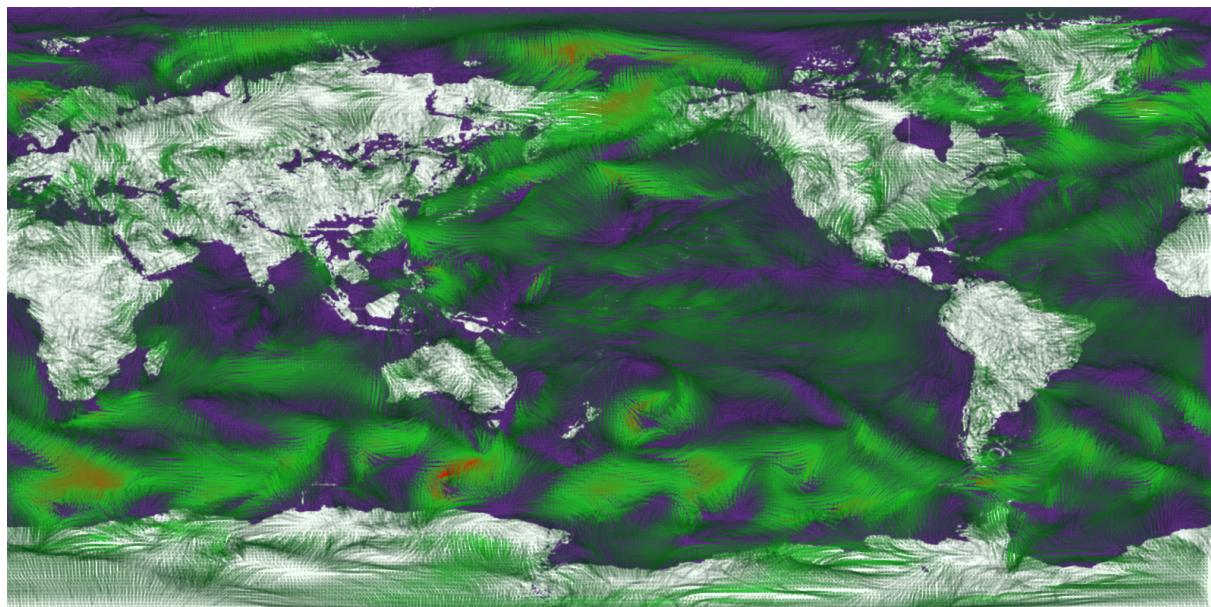


Figura 12 - Simulación de Test5

C A M B I O S P R I N C I P A L E S

- a. Generación de una curva por medio de bezier()
- b. Mapa estático
 - i. Descarte de la función de Zoom (problema en el manejo de capas)
- c. Colores en degradado y ajuste de transparencia (interpolación para colores)

```
// En la clase Particle
PVector bezierControl;
PVector prevPos;
void update() {
    vel.add(acc);
    vel.limit(maxSpeed);
    pos.add(vel);
    int k = 15;
    bezierControl.set(pos.x + vel.x * k, pos.y + vel.y * k);
```

```

prevPos = pos.copy();

void show() {
    stroke(particleColor[0], particleColor[1], particleColor[2],
particleColor[3]);
    strokeWeight(3);
    bezier(prevPos.x, prevPos.y, bezierControl.x, bezierControl.y,
        bezierControl.x, bezierControl.y, pos.x, pos.y);

// Interpolación de colores
void updateColor(float mag) {
    float normalizedMag = map(mag, minMag, maxMag, 0.0, 1.0);

    if (normalizedMag < 0.5) {
        float t = map(normalizedMag, 0.0, 0.5, 0.0, 1.0);
        particleColor[0] = lerp(0, 30, t);           // R: azul a verde
        particleColor[1] = lerp(0, 200, t);          // G
        particleColor[2] = lerp(10, 39, t);          // B

    } else {
        float t = map(normalizedMag, 0.5, 1.0, 0.0, 1.0);
        particleColor[0] = lerp(30, 255, t);         // R: verde a rojo
        particleColor[1] = lerp(200, 0, t);          // G
        particleColor[2] = lerp(39, 0, t);          // B
    }
    particleColor[3] = map(mag, minMag, maxMag, 0, 240); // Alpha
}

```

La generación de curvas Bezier crea trazos más fluidos para representar el viento, puesto que genera curvas para visualizar el movimiento. Esto se genera por medio de dos puntos (inicial y final) y por una serie de puntos intermedios dados por la constante de Bezier. Por otro lado, el mapa estático (descarte de la función de desplazamiento) resuelve problemas de manejo de capas. La interpolación de colores en degradado con ajuste de transparencia permite visualizar la intensidad del viento de forma más precisa, y evita que se sature la vista del usuario con demasiadas partículas. Sin embargo, el movimiento no es tan fluido como en la versión anterior.

5. Test6 - Mapa por actualización, zoom y desplazamiento

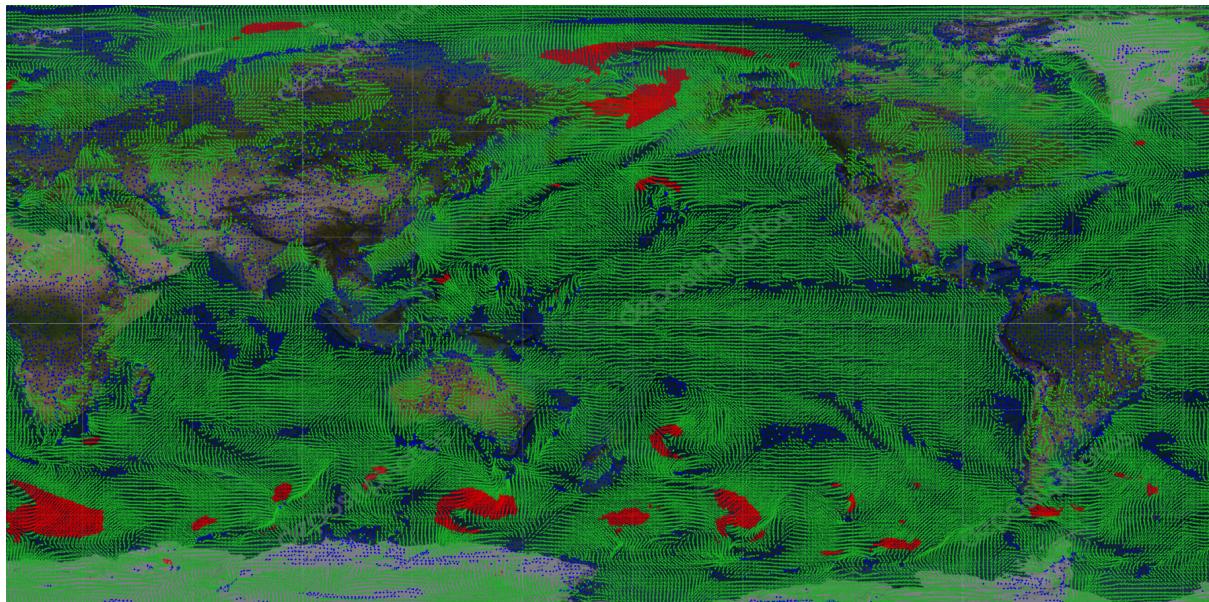


Figura 13 - Simulación de Test6

CAMBIOS PRINCIPALES

- Restauración de las funciones de zoom y desplazamiento
- Actualización del mapa por frame de actualización de partículas (renderizado continuo)

```
// Variables globales
float offX, offY;
float zoom = 1;
float minZoom = 1;
float maxZoom = 5;

void draw() {
    // Aplicar transformaciones
    translate(offX, offY);
    scale(zoom);
    translate(-width/2, -height/2);

    // Renderizado continuo
    image(earth, 0, 0);
    mapLayer.beginDraw();
    mapLayer.fill(0, 3);
    mapLayer.rect(0, 0, width, height);
    // ... actualización de partículas
```

```

mapLayer.endDraw();
image(mapLayer, 0, 0);

// Control de zoom y desplazamiento
void mouseWheel(MouseEvent event) {
    zoom -= event.getCount() * 0.2;
    zoom = constrain(zoom, minZoom, maxZoom);
}
void mouseDragged() {
    offX += (mouseX - pmouseX) / zoom;
    offY += (mouseY - pmouseY) / zoom;
}

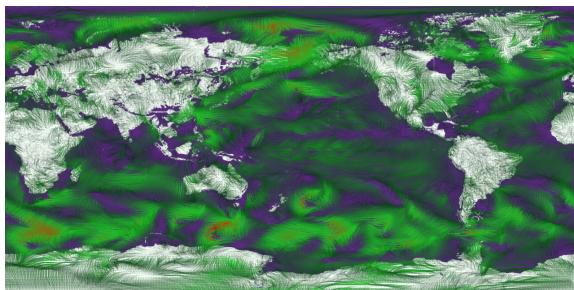
```

La restauración del zoom y desplazamiento mejora la interactividad permitiendo explorar áreas específicas del mapa, mientras que el renderizado continuo (actualización por frame) mantiene la fluidez de la animación. El sistema actualiza la posición de las partículas y redibuja el mapa en cada frame, evitando problemas de superposición de capas.

4. Resultados Obtenidos

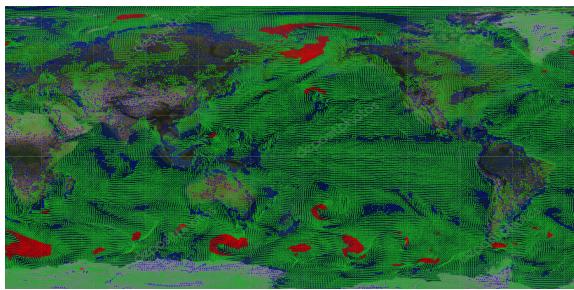


Test4



Disminución del ruido visual producto de graficar todas las partículas, y visualización de las corrientes más significativas

Test5



Ubicación geográfica clara dada por el mapa, y visualización de la fluidez en el movimiento de partículas

Test6

5. Conclusiones

Este proyecto logró satisfactoriamente cumplir con las expectativas del equipo, y se confía en que también cumplirá con las expectativas del profesor. Este proyecto supuso no solo un inmenso escalón hacia el mundo de la programación gráfica, sino que también significó la llave que permitió visualizar de manera práctica cómo generar una conexión entre la programación, los fenómenos del mundo físico, los principios matemáticos intrínsecos en ellos y la belleza del arte.

Por medio de este proyecto, se hicieron notorias las aplicaciones de la interpolación bilineal, tanto para representar las fuerzas de los vectores, como para generar degradados de colores. Fue un primer acercamiento al manejo de capas en simulaciones gráficas y también, para la comprensión de fenómenos meteorológicos y su forma tradicional de registro de datos.

Como cualquier cosa en la vida, este proyecto tiene áreas de mejora y extensiones futuras. Podría implementarse una optimización del algoritmo para leer datos más grandes, o incluir una lectura y graficación de datos en tiempo real. Por su parte, es escalable para otro tipo de corrientes, o fenómenos meteorológicos, por lo que se podrían incluir datos de corrientes marítimas, o capas de temperatura y presión atmosférica.

6. Apéndices

1. Funciones básicas en Processing

Renderizado de gráficos

1. Funciones básicas de dibujo:

- `line(x1, y1, x2, y2)` - Dibuja líneas.
- `rect(x, y, width, height)` - Dibuja rectángulos.
- `ellipse(x, y, width, height)` - Dibuja elipses o círculos.
- `triangle(x1, y1, x2, y2, x3, y3)` - Dibuja triángulos.

2. Ajuste de colores y estilos:

- `fill(r, g, b)` - Colorea el interior de una figura.
- `stroke(r, g, b)` - Define el color del borde.
- `noFill()` / `noStroke()` - Quita el relleno o el borde.

3. Opciones de renderizado avanzado:

- `size(width, height, P2D)` o `size(width, height, P3D)` - Configura el motor 2D o 3D.
 - `beginShape()` y `endShape()` - Crea formas complejas personalizadas.
-

Manejo de eventos de usuario

1. Eventos del mouse:

- `mousePressed()` - Detecta cuándo se presiona un botón del mouse.
- `mouseReleased()` - Detecta cuándo se suelta el mouse.
- `mouseMoved()` / `mouseDragged()` - Detecta movimientos del mouse.

2. Eventos del teclado:

- `keyPressed()` - Detecta cuando se presiona una tecla.
- `keyReleased()` - Detecta cuando se suelta una tecla.
- `key` y `keyCode` - Identifican la tecla presionada.

3. Variables globales del mouse y teclado:

- `mouseX, mouseY` - Coordenadas actuales del mouse.
 - `pmouseX, pmouseY` - Coordenadas previas del mouse.
-

Sistema de partículas

1. Clases y estructuras básicas:

- Define una clase `Particle` con atributos como posición, velocidad y color.
 - Crea un array o `ArrayList` de partículas.
2. **Funciones para actualizar partículas:**
 - `update()` - Cambia la posición según la velocidad.
 - `display()` - Dibuja cada partícula en pantalla.
 - `applyForce(force)` - Simula fuerzas como gravedad o viento.
 3. **Integración con funciones nativas:**
 - `random()` - Genera valores aleatorios para movimiento inicial.
 - `lerp()` - Interpola entre posiciones para efectos suaves.
-

Animación en tiempo real

1. **Ciclo principal de animación:**
 - `draw()` - Se ejecuta continuamente para actualizar la pantalla.
2. **Control de velocidad:**
 - `frameRate(fps)` - Establece la velocidad de actualización.
3. **Actualización dinámica:**
 - Variables para posiciones y movimientos que se actualizan en cada iteración.
 - `background()` para limpiar y evitar trazos antiguos.
4. **Interpolación para suavidad:**
 - `lerpColor()` - Suaviza la transición entre colores.
 - `sin()` y `cos()` - Animaciones oscilatorias.

2. Repositorio del proyecto en GitHub

<https://github.com/TGMAPA/windSim>

Fuentes de Información Utilizadas

1. *Processing Foundation.* (2024). Processing Reference.
2. *Stack Overflow.* (n.d.). Bilinear interpolation using Java.
3. *GeoMap Library Documentation*
4. *JSON.* (n.d.). <https://www.json.org/json-es.html>

Enlaces a Bibliotecas o Documentación Externa

1. *Processing*: <https://processing.org/reference/>
2. *GeoMap*: <https://www.gicentre.net/geomap/>
3. *JSON Processing*: <https://processing.org/reference/libraries/>