

Système à particules pour la visualisation de données spatio-temporelles

Projet de programmation 2

Marguery Jules
Mateusz Birembaut

L3 informatique
Faculté des Sciences
Université de Montpellier

2023-2024

Résumé

Ce rapport décrit la conception et le développement de systèmes à particules simples, puis l'implémentation de systèmes pour visualiser des données spatio-temporelles, et enfin, l'évaluation des performances des systèmes développés.

Table des matières

1	Introduction	3
1.1	Systèmes à particules	3
1.1.1	Définition	3
1.2	Les systèmes à particules dans Canvas	3
1.3	earth : :nullschool	4
1.4	Enjeux du projet	4
2	Développements Logiciel : Conception, Modélisation, Implémentation	5
2.1	Conception	5
2.1.1	Technologies Utilisées	5
2.1.2	Développement et Implémentation	5
2.1.3	Système "Physic"	6
2.1.4	Système "Fire"	6
2.1.5	Système "ElectionMap"	7
2.1.6	Système "Windmap"	7
2.1.7	Génération du fond de la carte des vents	8
2.1.8	Système "Tree"	9
2.1.9	Système "Firework"	10
2.2	Modélisation	10
2.2.1	Carte des vents	10
2.3	Fonctionnalités	13
2.4	Format de données	14
2.4.1	Carte des vents	14
2.4.2	Carte des élections	15
2.5	Statistiques	15
3	Algorithmes et Structures de Données	16
3.1	Structures de données principales	16
3.1.1	Particle	16
3.1.2	Données spatiotemporelles	16
3.1.3	Grille de vecteur	17
3.2	Algorithmes principaux	18
3.2.1	Complexité	18
4	Analyse des résultats	19
4.1	La classe Tester	19
4.2	Résultats des tests	19
4.2.1	Physic et windmap	19
4.2.2	Firework	20
4.3	Onglet Performance	21
5	Gestion du Projet	22
5.1	Organisation	22
5.2	Versioning, Utilisation de GitHub	22
5.3	Changements majeurs	22
6	Bilan et Conclusions	23
6.1	Perspectives d'amélioration ²	23
6.2	Conclusion	23
7	Annexe	25

1 Introduction

1.1 Systèmes à particules

Nous assistons à une explosion du nombre de données, notamment des données spatio-temporelles dans divers domaines comme la météorologie, l'urbanisme, ou encore la biologie, la nécessité d'avoir des outils pour leur visualisation et leur analyse devient cruciale. Les systèmes à particules sont particulièrement adaptés pour simuler des phénomènes complexes et dynamiques se basant sur ce type de données.

1.1.1 Définition

Les systèmes à particules sont définis pour la première fois par Hubert Reeves [5] qui les utilise pour représenter des "fuzzy objects", qu'il décrit comme des objets qui ont une surface irrégulière, complexe et mal définie. Les systèmes à particules sont généralement définis par un groupe d'éléments primitifs (particules) qui suivent des règles qui définissent leur mouvement et leur apparence.

Un système de particules typique comprend plusieurs étapes clé pour simuler et afficher ces particules, en général découpé en 3 étapes (répétées plusieurs fois tout au long de l'exécution) :

- Étape d'émission : Création de Particules : on détermine combien de particules doivent être créées à chaque unité de temps. Chaque nouvelle particule est initialisée avec des propriétés spécifiques, telles que leur position initiale, vitesse, et autres paramètres tels que la couleur, taille, et durée de vie.
- Initialisation des Propriétés : les propriétés initiales peuvent être influencées par l'émetteur, qui peut être statique, suivre une trajectoire, ou même réagir à des interactions dans l'environnement, comme le vent ou d'autres forces.
- Étape de simulation : Mise à jour des Particules : chaque particule est mise à jour selon les forces et interactions définies dans le système, telles que la gravité, ou des forces personnalisées.
- Gestion de la Durée de Vie : les particules qui ont dépassé leur durée de vie prévue sont retirées ou remise à leur position de départ.
- Étape de rendu : après chaque mise à jour, on affiche chaque particule.

1.2 Les systèmes à particules dans Canvas

Dans son article "How I built a wind map with WebGL" [1], Vladimir Agafonkin généralise la plupart des cartes des vents créée avec Canvas comme suit :

- On génère une liste de particules avec des positions aléatoire et on les dessine.
- Pour chaque particule, on récupère la vitesse du vent à sa position, et on déplace la particule en fonction.
- On repositionne une partie des particules pour ne pas avoir de zone de vide.

- On rend l'image actuelle transparente et on dessine les nouvelles particules par-dessus.

Nous nous sommes inspirés de cette généralisation pour le design de nos systèmes à particule, et plus particulièrement notre carte des vents.

1.3 earth : :nullschool

La carte des vents de Cameron Beccario est une source d'inspiration majeure pour notre carte des vents, elle est disponible à l'adresse <https://earth.nullschool.net>.

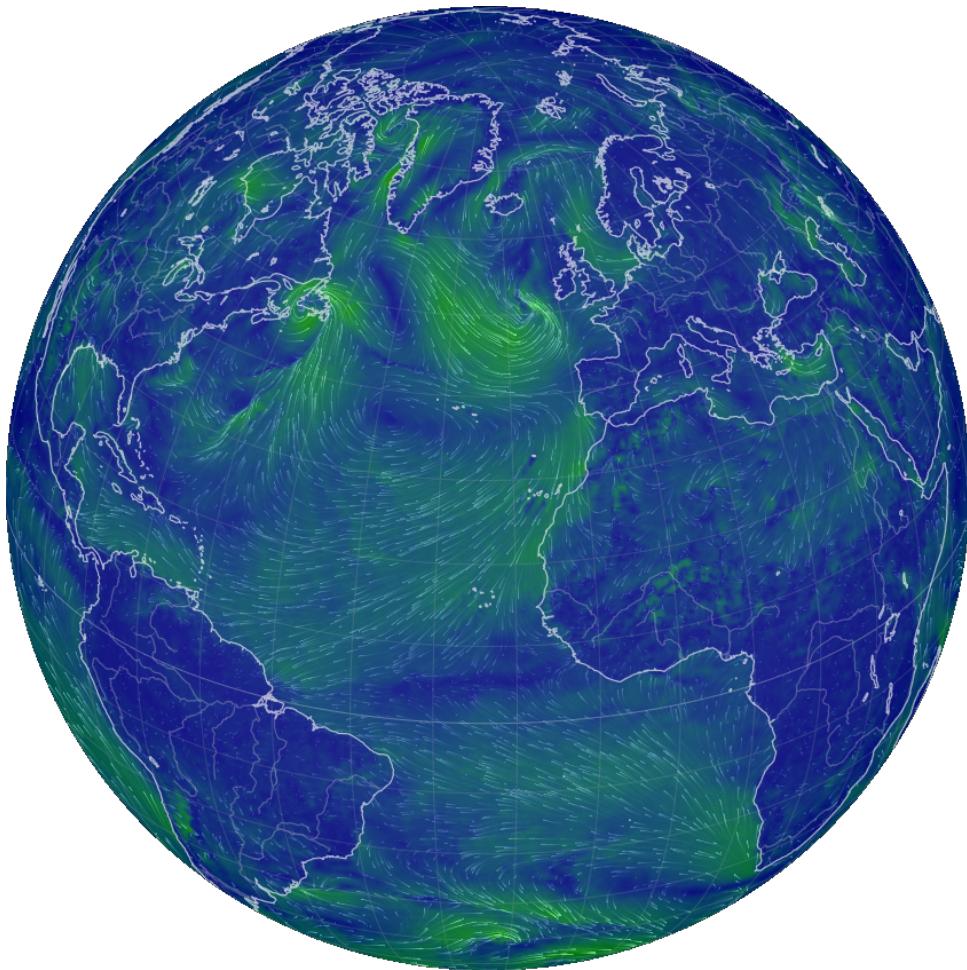


FIGURE 1 – Visualisation des vents sur earth.nullschool.net

1.4 Enjeux du projet

C'est dans ce contexte que ce projet s'inscrit et vise donc à explorer l'utilisation des systèmes à particules. Dans un premier temps, cela consistera à se familiariser avec différents systèmes à particules simples, ensuite, utiliser l'expérience acquise pour créer d'autres systèmes à particules pour la visualisation des données spatio-temporelles, facilitant ainsi leur compréhension et leur exploitation.

2 Développements Logiciel : Conception, Modélisation, Implémentation

2.1 Conception

Le but principal de notre projet était de créer des outils visuels interactifs qui permettent aux utilisateurs de visualiser et de manipuler des simulations de particules en temps réel. L'utilisation de systèmes à particules vient de la nécessité de comprendre et de représenter graphiquement des phénomènes complexes tels que la visualisation des votes en France (modélisés par notre système "ElectionMap") ou bien la visualisation des flux d'air dans l'environnement (modélisé par notre système "WindMap").

Le projet a impliqué le développement de plusieurs systèmes à particules. Nous avons commencé à créer des systèmes à particules simples afin de nous familiariser avec cette notion. Les premiers visaient à simuler des phénomènes physiques simples puis nous en avons créé d'autres dans le but de visualiser des données spatio-temporelles de manière interactive. Ces systèmes ont été conçus en utilisant JavaScript et Canvas HTML.

2.1.1 Technologies Utilisées

Nous avons opté pour des technologies web modernes pour une intégration et une accessibilité accrues. JavaScript a été choisi pour sa flexibilité et son intégration avec le Canvas HTML, qui offre des capacités puissantes de dessin et de manipulation graphique. Ces technologies nous ont permis de développer des simulations dynamiques et interactives, directement accessibles via le navigateur web, sans nécessité d'installations logicielles supplémentaires pour les utilisateurs finaux.

2.1.2 Développement et Implémentation

Le développement a été structuré autour de plusieurs itérations, chacune visant à intégrer de nouvelles fonctionnalités et à affiner les systèmes existants en fonction des retours obtenus lors des phases de test. Les principaux défis rencontrés sur notre carte représentant les élections étaient la mise en forme des données. Sur notre carte des vents, cela concernait la gestion de la performance, contraint par les capacités d'exécution des navigateurs.

2.1.3 Système "Physic"

Ces projets plus simples servaient d'exercices préparatoires, où des particules étaient soumises à des lois physiques de base comme la gravité et la conservation de l'énergie lors des collisions.

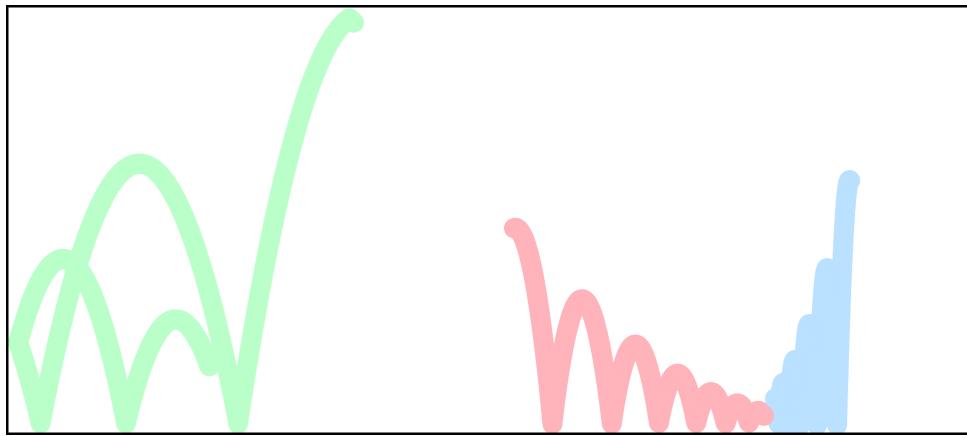


FIGURE 2 – 3 Particules qui simulent la gravité et qui rebondissent au contact d'un bord

Les particules ont toutes la même taille et force de rebond. Elles ont une couleur, une position de départ et une vitesse de base aléatoire. À chaque fois qu'on voudra déplacer une particule, on ajoute à son vecteur position son vecteur vitesse, et on soustrait les vecteurs forces qu'on veut appliquer (dans notre cas, la gravité uniquement).

2.1.4 Système "Fire"

Ce système à particules vise à reproduire le comportement d'un feu. Les particules de feu sont générées à partir d'un point ou d'une zone définie grâce au clic droit. Chaque particule a des paramètres modifiables grâce au menu, elles s'élèvent et se dissipent graduellement.

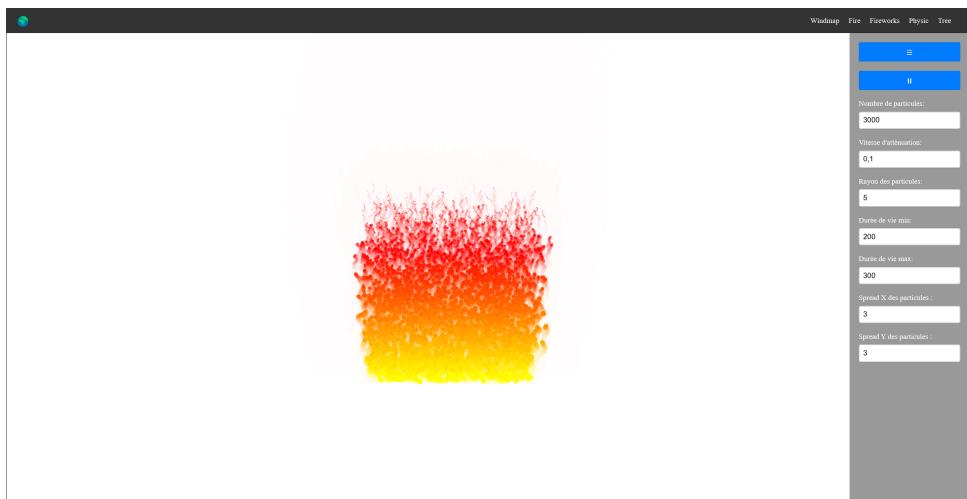


FIGURE 3 – Image du système simulant un feu

2.1.5 Système "ElectionMap"

Ce système utilise les données spatio-temporelles sur les votes des présidentielles en France. Chaque particule a une taille définie en fonction du nombre de votants, une couleur en fonction du parti et une position en fonction de la ville.

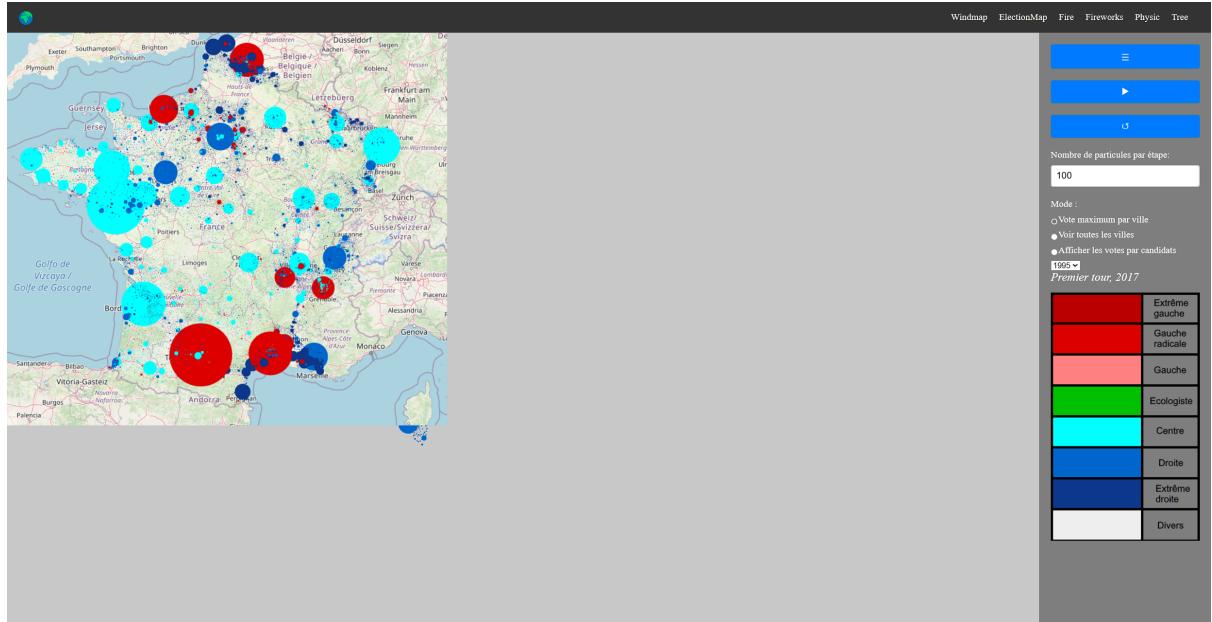


FIGURE 4 – Carte des élections

Les données utilisées sont celles des élections présidentielles du premier et second tour de 1995 à 2022 [3]. À chaque tour, les statistiques de votes de chaque ville sont enregistrées (comme le pourcentage de voix pour chaque candidat, le nombre total de votes, etc.), le code INSEE est mis en relation avec un tableau contenant les coordonnées latitudes et longitude et chaque ville.

Il est possible de voir le candidat majoritaire de chaque ville, la proportion de vote par ville pour chaque candidat d'une année donnée et la position de toutes les villes de France. Les couleurs choisies sont celles présentent sur la page Wikipédia des Élections présidentielles en France. [6]

2.1.6 Système "Windmap"

Ce système simule des flux d'air sur une carte géographique, utilisant des données météorologiques qui sont stockées dans une grille de vecteurs pour créer le fond et aussi diriger le mouvement des particules. Les interactions utilisateur permettent de modifier les paramètres de simulation comme la vitesse de déplacement et la densité des particules.

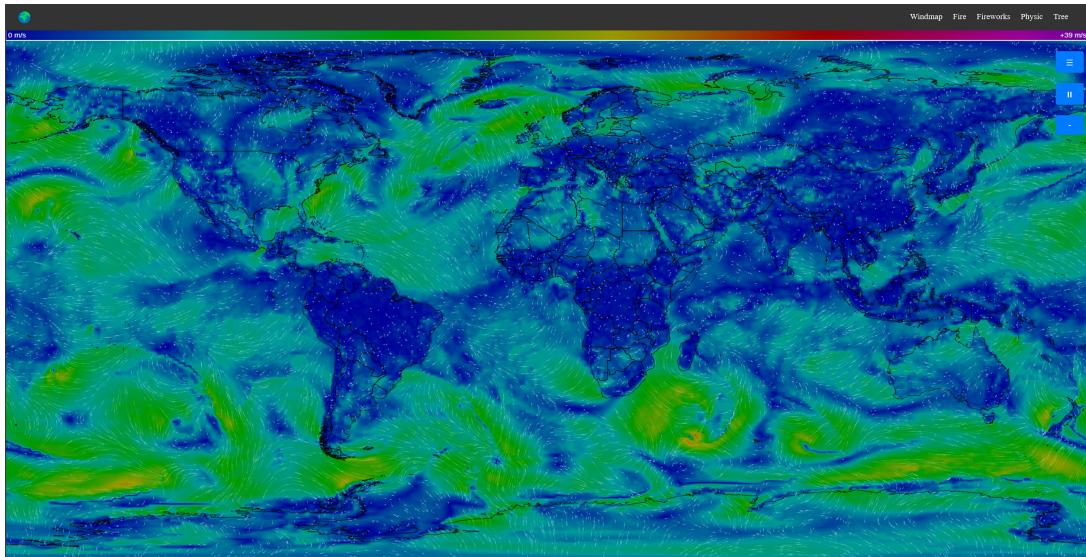


FIGURE 5 – Carte des vents

2.1.7 Génération du fond de la carte des vents

On crée une image de fond en parcourant chaque pixel. Comme nos données sont disponibles uniquement à des intervalles espacés, on utilise une interpolation bilinéaire. Pour chaque pixel, on récupère les quatre vitesses du vent les plus proches, en prenant en compte la distance entre chacun de ces points et en faisant la moyenne, on obtient une vitesse qui nous sert ensuite à colorer notre pixel. Cela permet de réduire la pixellisation pour le fond. On utilise aussi cette interpolation pour le déplacement des particules qui permet un déplacement plus précis et naturel.

Enfin, nous avons utilisé d3js, une librairie JavaScript qui permet facilement de créer les contours des pays en fournissant un fichier et spécifiant une projection, ici cylindrique équidistante. Cette projection représente la terre en conservant les latitudes et les longitudes sous forme de lignes droites qui forment un quadrillage, ce qui est particulièrement utile à nos données météorologiques. Elle nous permet également de confirmer que nos données sont correctement affichées, les continents sur la carte de fond sont visibles sans le contour des pays, ce qui rend visible tout décalage.

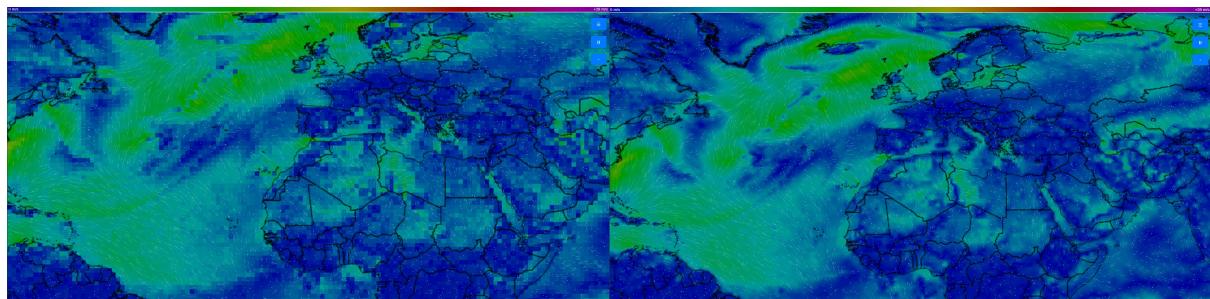


FIGURE 6 – Fond sans interpolation / Avec interpolation

Pour l'échelle de couleurs, nous avons fait quelques recherches, nous avons regardé différentes implémentations de windmap, comme celle de Vladimir Agafonkine [1], ou encore de Cameron Beccario[2]. Nous avons finalement choisi de nous rapprocher de l'échelle d'un outil de visualisation de fichiers météorologiques grib2 "XyGrib".

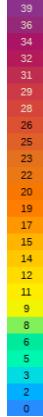


FIGURE 7 – Echelle XyGrib

2.1.8 Système "Tree"

Ce système simule la pousse d'un arbre avec un seul type de particule.

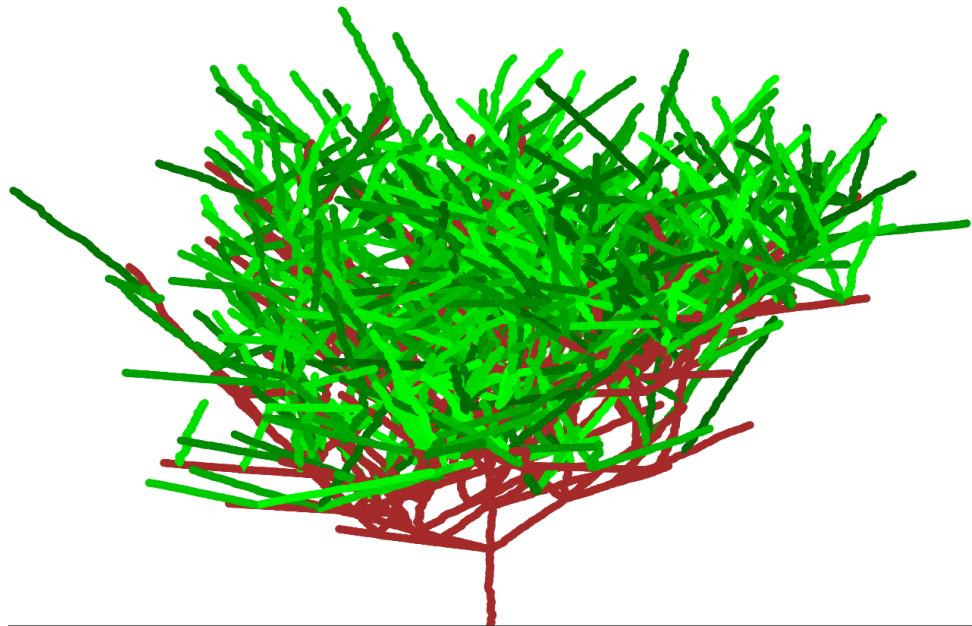


FIGURE 8 – Exemple de génération du système Arbre

Une seule particule est générée à l'origine. Au bout d'un nombre d'images, on essaie de générer des particules dans des directions aléatoires. Il est possible de spécifier le nombre de particules maximum qu'on veut générer, la chance que chaque particule se génère et la hauteur maximale à atteindre avant d'arrêter de créer des particules.

2.1.9 Système "Firework"

Ce système simule des feux d'artifice avec des particules "Firework" et des particules "Physics". Il est possible de spécifier le nombre de particules "physics" qui seront créées à la fin de la vie de la particule "Firework" de base.



FIGURE 9 – Ancien système de trace

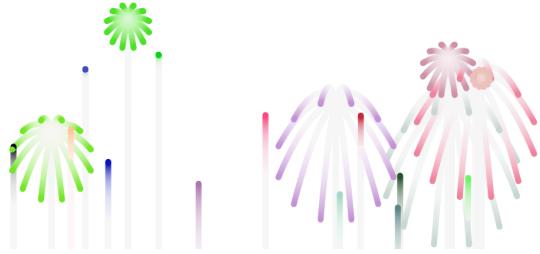


FIGURE 10 – Nouveau système de trace

Nous avons décidé de retravailler notre méthode pour l'affichage d'une trace derrière les particules en s'inspirant de la définition de Vladimir Agafonkin [1]. Notre ancien système gardait les dix dernières positions de chaque particule et dessinait des cercles de plus en plus petits en fonction de l'âge de la position, tandis que le nouveau système ajoute simplement une image semi-transparente par-dessus toutes les particules, donnant une illusion de trace.

La nouvelle méthode est bien plus économique en calcul, mais permet moins de manipulations sur la trace, comme des longueurs ou tailles différentes selon les particules.

2.2 Modélisation

2.2.1 Carte des vents

Nous avons utilisé de nombreuses variables et méthodes statiques pour ce projet étant donné que la plupart de nos classes n'auraient eu besoin que d'une seule instance pour faire fonctionner notre carte.

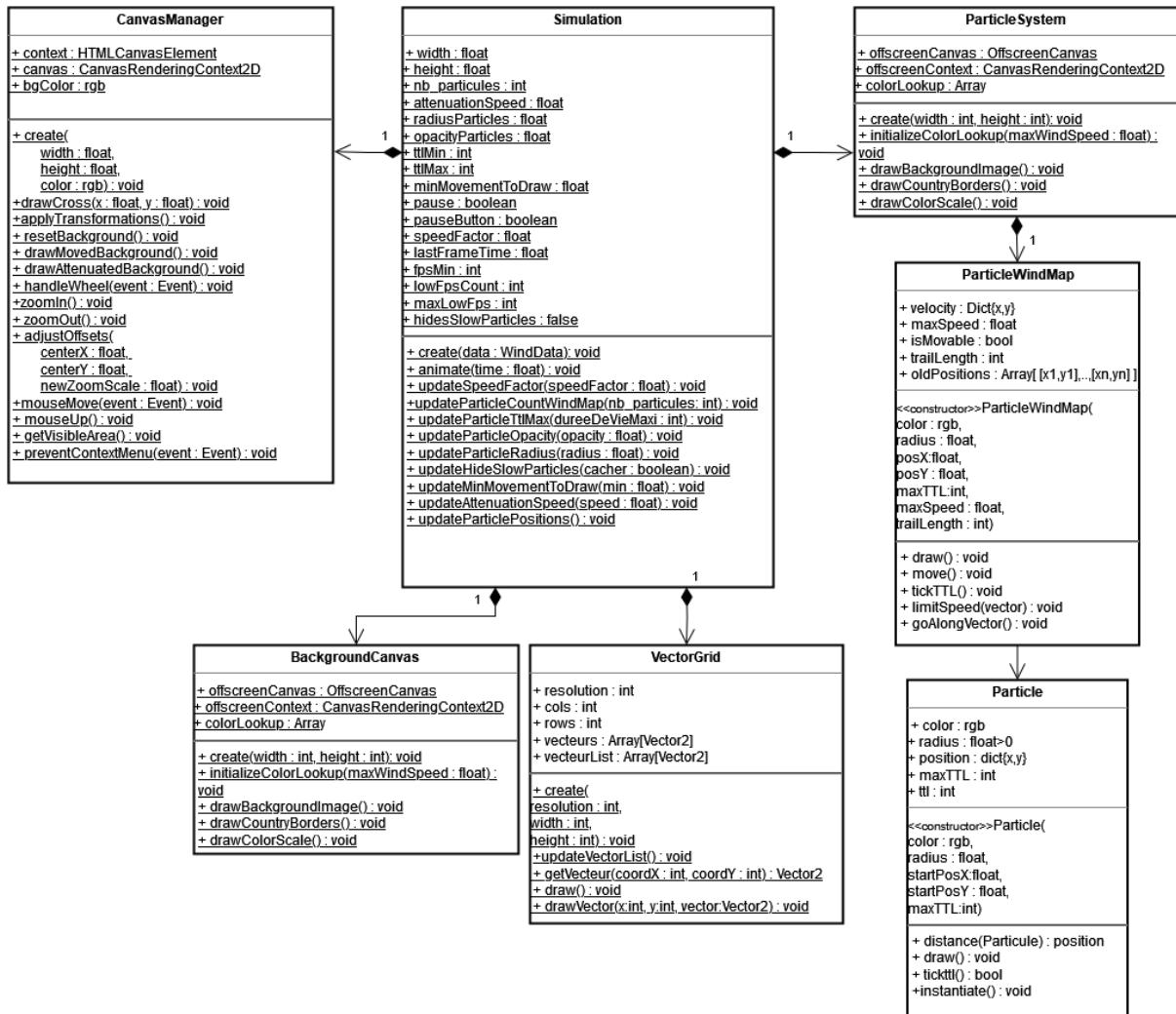


FIGURE 11 – Diagramme de classe de la carte des vents

1. CanvasManager :
 - Fonctionnalité : gère le rendu des particules sur le canvas, en prenant en compte les transformations telles que le zoom et le déplacement.
 - Responsabilités : initialisation du canevas, gestion des événements de souris pour le zoom, le déplacement et rafraîchissement du canvas lors des mises à jour de la simulation.
2. ParticleSystem :
 - Fonctionnalité : gère la création, la mise à jour, et la gestion de la durée de vie des particules.
 - Responsabilités : génération des particules basée sur des paramètres dynamiques et gestion de l'extinction des particules.
3. VectorGrid :
 - Fonctionnalité : stocke les vecteurs représentant les forces qui seront appliquées aux particules du système "WindMap".
 - Responsabilités : interpolation des données de vecteur pour fournir des forces continues pour créer le fond du canvas et aux particules, le calcul des vecteurs

est basé sur les coordonnées x, y.

4. Particle (et sous-classes telles que ParticlePhysic et ParticleWindMap) :
 - Fonctionnalité : représente les particules individuelles dans la simulation qui se déplacent et interagissent selon les différents systèmes.
 - Responsabilités : mise à jour des positions et des vitesses basées sur des forces externes, rendu graphique des particules, et gestion de la collision et du rebond.
 5. Simulation :
 - Fonctionnalité : coordination globale du cycle de vie de la simulation, de l'initialisation à l'exécution continue.
 - Responsabilités : contrôle des états de simulation, gestion des performances, ajustements dynamiques des paramètres de la simulation en fonction des utilisateurs.
- La simulation commence par l'initialisation de la classe **Simulation**, on fait un appel à la méthode statique `create(data)`, où `data` correspond à un dictionnaire de données contenant les données météorologiques relatives aux vents. Elle initialise ensuite les classes **CanvasManager**, **BackgroundManager**, **VectorGrid** et **ParticleSystem**.

Pour le site web, nous avons créé un diagramme cas utilisation pour résumer les actions disponibles, nous détaillerons quelques fonctionnalités dans la prochaine section.

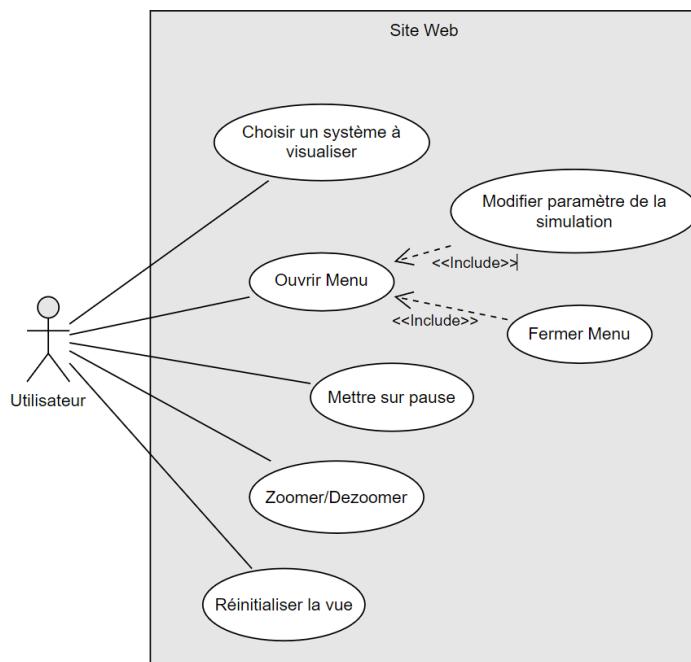


FIGURE 12 – Diagramme de cas d'utilisation

2.3 Fonctionnalités

Afin de visualiser nos systèmes à particules, nous avons décidé de développer un site web.

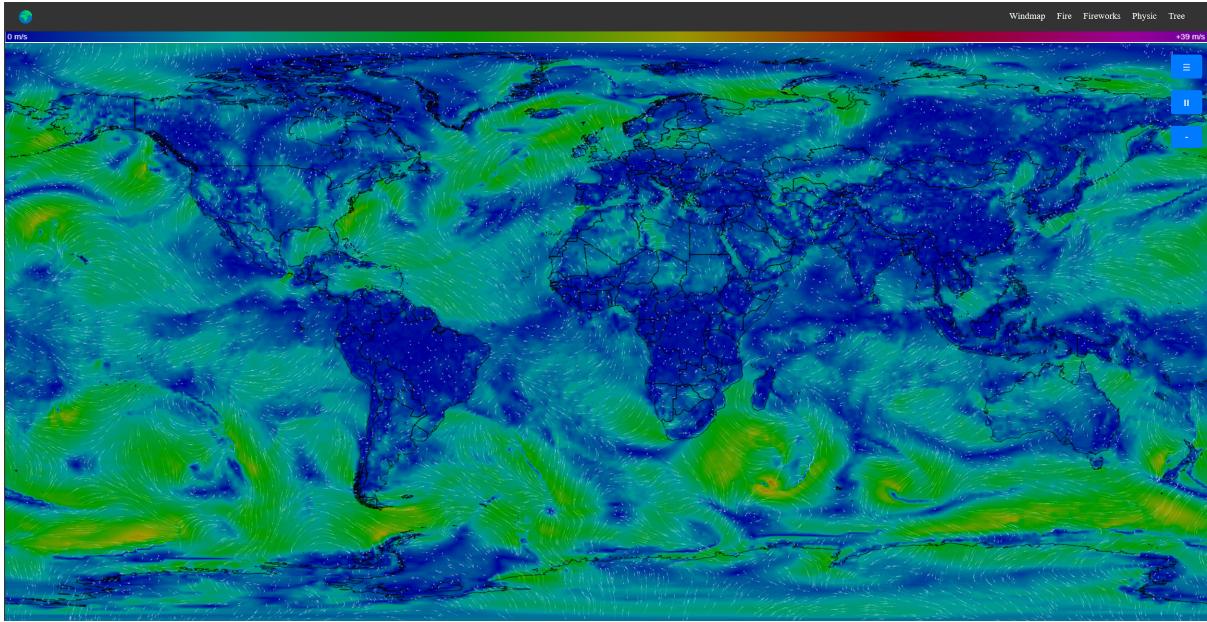


FIGURE 13 – Site web avec nos différents systèmes à particules

La barre en haut permet d'accéder aux différents systèmes, et les 3 boutons en haut à gauche permettent respectivement de :

- Accéder au menu de modification.
- Mettre en pause la simulation.
- Remettre le zoom à zéro et recentrer la carte (uniquement pour la carte des vents).

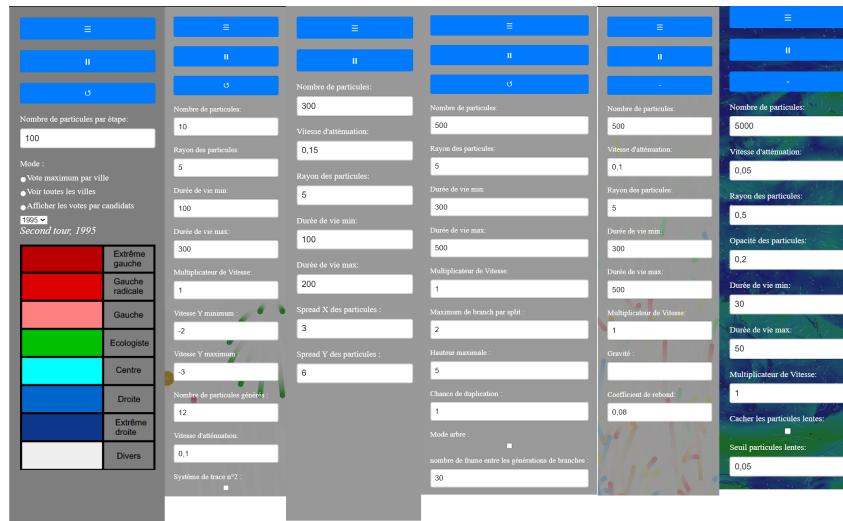


FIGURE 14 – Menus des différents systèmes à particules

Le menu de modification nous permet de modifier certaines variables ou aspects visuels de chaque système à particules, comme le nombre de particules que l'on souhaite afficher, de cacher les particules qui vont moins vite qu'un certain seuil ou leur taille, d'afficher le nombre de votes de chaque candidat lors d'une élection.

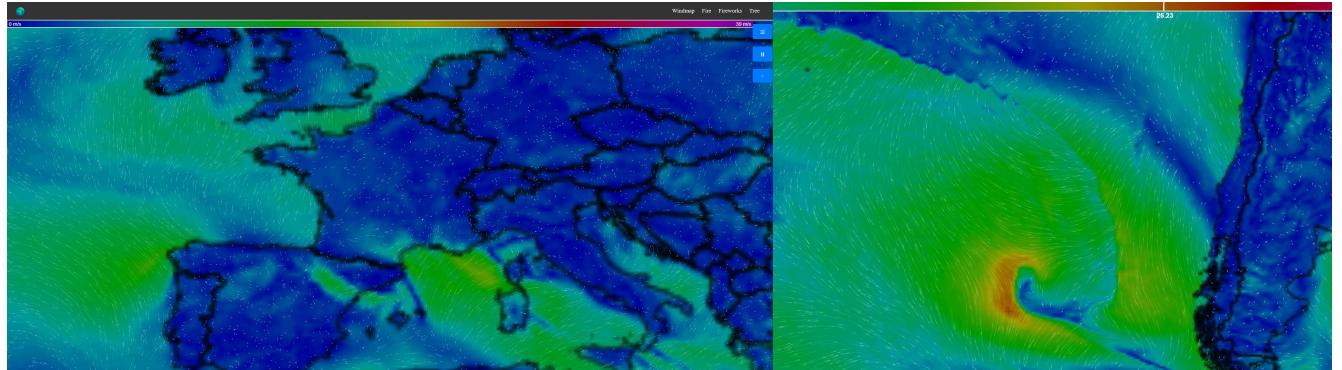


FIGURE 15 – Zoom / Clic droit

Pour la carte des vents, nous avons ajouté la possibilité de zoomer et de dézoomer sur notre carte ainsi que d'utiliser le clic droit afin de connaître la vitesse des vents à l'endroit où l'on clique. On peut aussi déplacer la vue lorsque l'on reste appuyé sur le clic gauche et que l'on fait glisser la souris.

2.4 Format de données

2.4.1 Carte des vents

Les données spatio-temporelles que nous utilisons pour la carte des vents nous proviennent du Global Forecast System (GFS) du National Center for Environmental Prediction (NCEP) [4] :

- Le GFS est un système de prédition météorologique qui s'appuie sur de nombreux facteurs (comme la pression, l'humidité et les vents) pour tenter de prédire la météo.
- Il propose des prédictions toutes les six heures et sur les seize jours suivants par tranches de trois heures.
- On peut filtrer les prédictions par niveau (par exemple 1000 m au-dessus du sol) et par type de données (par exemple la température) pour ne télécharger que les informations qui nous sont utiles.

Nous récupérons les composantes U et V du vent (respectivement la vitesse horizontale et verticale du vent) en mètres par seconde à dix mètres au-dessus du sol. Les données sont sous la forme de deux tableaux de flottants. Chaque tableau contient également un header avec une composante nX et nY qui nous permet de déterminer la taille de notre matrice de vecteur.

2.4.2 Carte des élections

Les données utilisées pour la carte des élections nous proviennent de data.gouv.fr [3]. Nous avons besoin de plusieurs jeux de données : un qui contient les votes détaillés pour chaque ville, et un qui contient les positions latitude/longitude de chaque ville de France ainsi que leur code unique INSEE. Nous utilisons les données des premiers et seconds tours des élections de 1994 à 2022.

Certaines informations, comme les votes blancs, n'étaient pas enregistrées en 1994, les données de votes ont dû être nettoyées préalablement à leur utilisation. Une table contenant le nom de tous les candidats et leur courant politique a été créée, pour les mettre en relation pour l'affichage.

Les positions des villes étant donnés en coordonnées géographiques, un calcul est nécessaire pour avoir la position x, y de chaque ville sur le canvas :

```
\PositionVille = {
    x : (longitude + offsetX)*echelleX,
    y : (offsetY - latitude) * echelleY
}
```

avec :

- **echelleX,echelleY**, l'échelle de notre image, calculée en divisant la taille de l'image à une valeur fixe, permettant d'avoir une mise à l'échelle convenable.
- **offsetX,offsetY**, correspondants aux coordonnées géographiques du coin supérieur gauche de notre image.

Une fois la position calculée, on peut mettre en relation le code Insee unique de chaque ville pour accéder à sa position sur le Canvas.

2.5 Statistiques

Nous avons écrit un total de 46 classes, réparties sur un total de 9 simulations, dont 3 étant des tests de performance (soit 6 systèmes à particules).

Type	Nombre de fichiers	Lignes vides	Lignes commentées	Lignes de code	Total
JavaScript	56	727	322	3307	4356
CSS	10	161	24	700	885
EJS	12	46	0	635	681
Total	78	934	346	4642	5922

3 Algorithmes et Structures de Données

3.1 Structures de données principales

Dans cette section, nous aborderons en détail les principales structures de données mises en place pour nos différents systèmes à particules et la représentation des données spatio-temporelles.

3.1.1 Particle

La classe Particle sert de base à toutes nos autres particules. Elle représente une particule ronde, immobile et qui disparaît après un temps donné. Voici comment elle est représentée :

```
// Particle
Particle = {
    Couleur : RGB,
    Rayon : float>0,
    Position : Vecteur,
    TempsDeVie : int,
    TempsÉcoulé : int
}
```

Le type Vecteur est un dictionnaire {x,y} utilisé pour représenter et faire des calculs entre des vecteurs à deux dimensions. Nous utiliserons la même structure pour tous les vecteurs du projet.

Particle existe principalement dans le but d'être spécialisé, par exemple l'ajout d'un attribut Vélocité à notre structure, et d'une méthode à notre classe, nous permet de mettre en place des particules qui bougent. Cela signifie aussi que nous aurons, pour la plupart des cas, une classe particule différente pour chaque système qu'on veut représenter.

3.1.2 Données spatiotemporelles

Une fois les données GFS transformé en JSON à l'aide de "Grib2Json", le résultat est un dictionnaire avec un header qui contient les spécifications des données, comme l'unité utilisée ou l'intitulé des données et un tableau de valeurs. Nous n'utilisons pas toutes les données du header pour la représentation du vent, mais quelques-unes sont importantes à souligner :

- **nX,nY** : Donne le nombre de vecteurs vents sur l'axe vertical et horizontal.
- **refTime** : Date et heure de l'émission de la prédiction.
- **forecastTime** : Nombre d'heures à ajouter à la date de l'émission de la prédiction.
- **{lo1,lo2} , {la1,la2}** : Point longitude/latitude des extrémités du modèle.
(Pour NFS, on aura toujours : {0,360} , {90,-90}.)

Dans le cas de la vitesse du vent, nous récupérons deux tableaux de flottants : un qui contient les valeurs de vitesse en mètres par seconde pour l'axe vertical et l'autre pour l'axe horizontal.

3.1.3 Grille de vecteur

La grille de vecteur est composée comme suit :

```
\VectorGrid
vectorGrid = {
    cols : int,
    rows : int,
    vecteurs : Array[Vecteur],
    maxWindSpeed : float,
    minWindSpeed : float
}
```

`cols`, `rows` correspondent à `nX` , `nY` vu précédemment. Le tableau `vecteurs` contient les vecteurs vitesses qu'on appliquera à chaque image aux particules pour représenter le vent sur la carte.

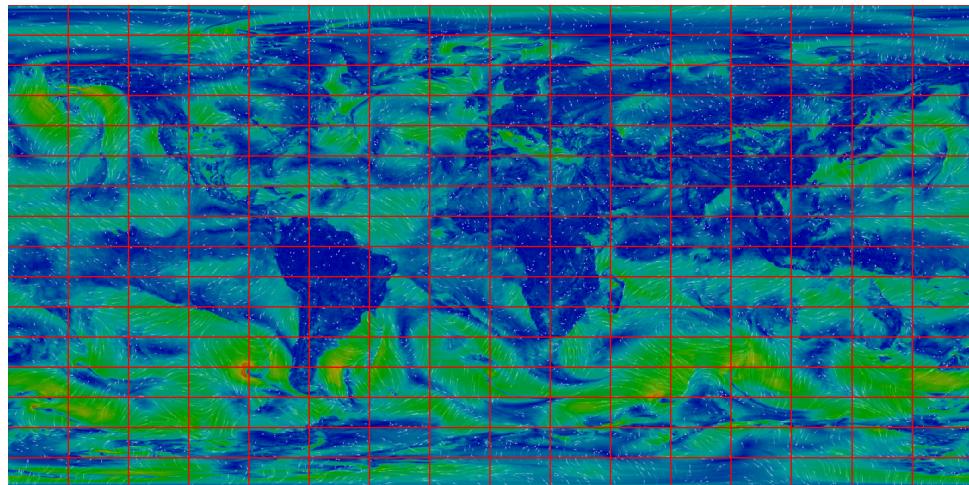


FIGURE 16 – Représentation visuelle de VectorGrid pour $nX = nY = 16$

3.2 Algorithmes principaux

```
ALGORITHME create() de Simulation pour une WindMap
DEBUT
    initialiserCanvas();
    vectorGrid = initialiserVectorGrid(fichier); // specifique a windmap
    dessinerFond(vectorGrid); // specifique a windmap
    creerParticleSystem(nb_particules);
    animer()

    animer() {
        tant que (1) {
            si (fps < 10 pendant 30 frames){
                nb_particules = nb_particules / 1.25
            }
            ajouterFondSemiTransparent();
            faireBougerParticules();
            dessinerParticules();
        }
    }
}

FIN
```

Cet algorithme est la structure principale de nos systèmes à particules, incluant l'initialisation de tout ce dont a besoin et la boucle d'animation.

3.2.1 Complexité

Dans le cas de la carte des vents, la complexité en dehors de la boucle "animate" est $O(n*m)$ où "n" est la largeur du canvas et "m" la hauteur car "dessinerFond(VectorGrid)" créer une image de taille $n*m$ soit deux boucles for imbriquées.

Dans la boucle animate, on a $O(n*m + p)$ où "p" est le nombre de particules, "ajouterFondSemiTransparent()" à la même complexité que "dessinerFond()", en plus, on dessine chaque particule qui a une complexité $O(p)$.

4 Analyse des résultats

4.1 La classe Tester

Nous avons développé une classe de test pour calculer les performances de nos systèmes à particules. On lui fournit un nombre de départs de particules, un nombre maximal, combien de particule est ce que l'on veut ajouter à chaque étape et le nombre de mesures qu'on veut faire par étape.

- Au début de chaque étape, Tester initialise un nouveau système à particule avec le nombre de particules souhaitées.
- À chaque seconde, on envoie à Tester le nombre d'images affichées.
- Une fois le nombre de mesures souhaitées atteint, on enregistre la moyenne. Si elle est inférieure à cinq images par seconde, on arrête le test.
- Une fois le test terminé, on écrit les données dans le fichier `\test\test.json`.

Plusieurs tests sont disponibles pour les systèmes physiques, feux d'artifices et la carte des vents aux adresses `\test\physic`, `\test\firework` et `\test\windmap`.

4.2 Résultats des tests

4.2.1 Physic et windmap

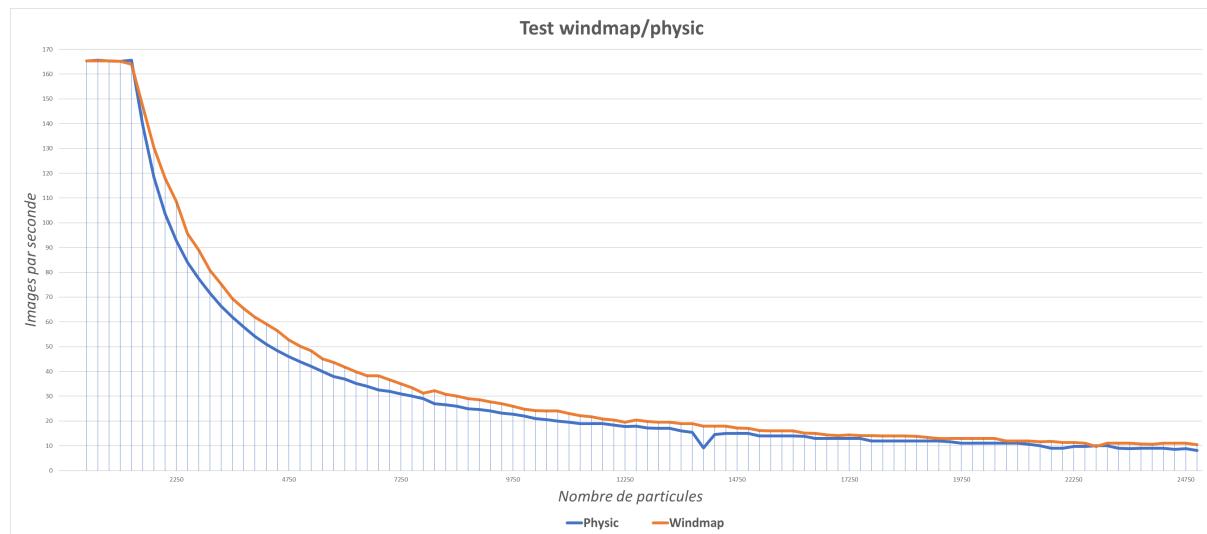


FIGURE 17 – Résultat des test physic

Les deux tests créés au maximum 25 000 particules. On remarque que les performances sont relativement similaires, avec environ 4 à 5 images par seconde de décalage. En effet, le calcul du déplacement est le même : on se contente d'ajouter un vecteur vitesse à un vecteur position, seul la structure de données change, et c'est principalement quand on met en place cette dernière (au démarrage) qu'on voit une différence : la simulation de particule physique se lance instantanément alors qu'il faut quelques secondes à la carte des vents avant de s'afficher.

4.2.2 Firework

Pour ces tests, nous allons jusqu'à un maximum de 5000 particules par pas de 50 particules et chaque particule se divise en 6 à la fin de sa vie.

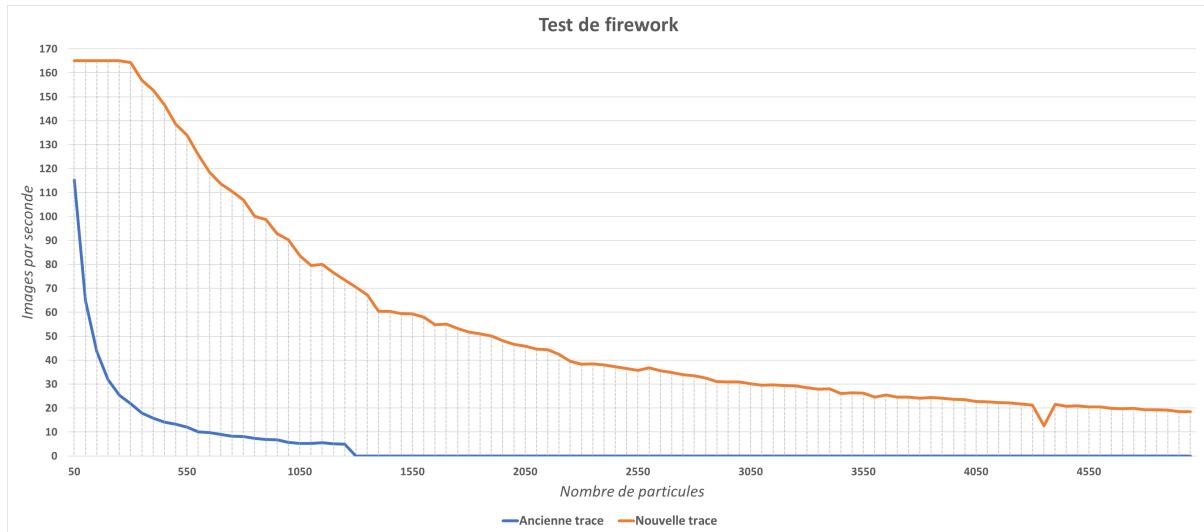


FIGURE 18 – Résultat des test des feux d'artifices

On remarque que l'ancien système de trace n'est capable de supporter que 1250 particules avant de descendre en dessous de 5 images par seconde. Cette différence s'explique par le fait qu'il demande 10 fois plus d'appels à `draw()`, qui est déjà une des fonctions responsables pour le plus de temps de calcul.

4.3 Onglet Performance

En plus de la classe de Tests, nous avons utilisé l'onglet "Performances" du navigateur pour avoir une idée d'où venait la perte de FPS. Voici les résultats pour 70 000 particules en enregistrant pendant 1,50 min.

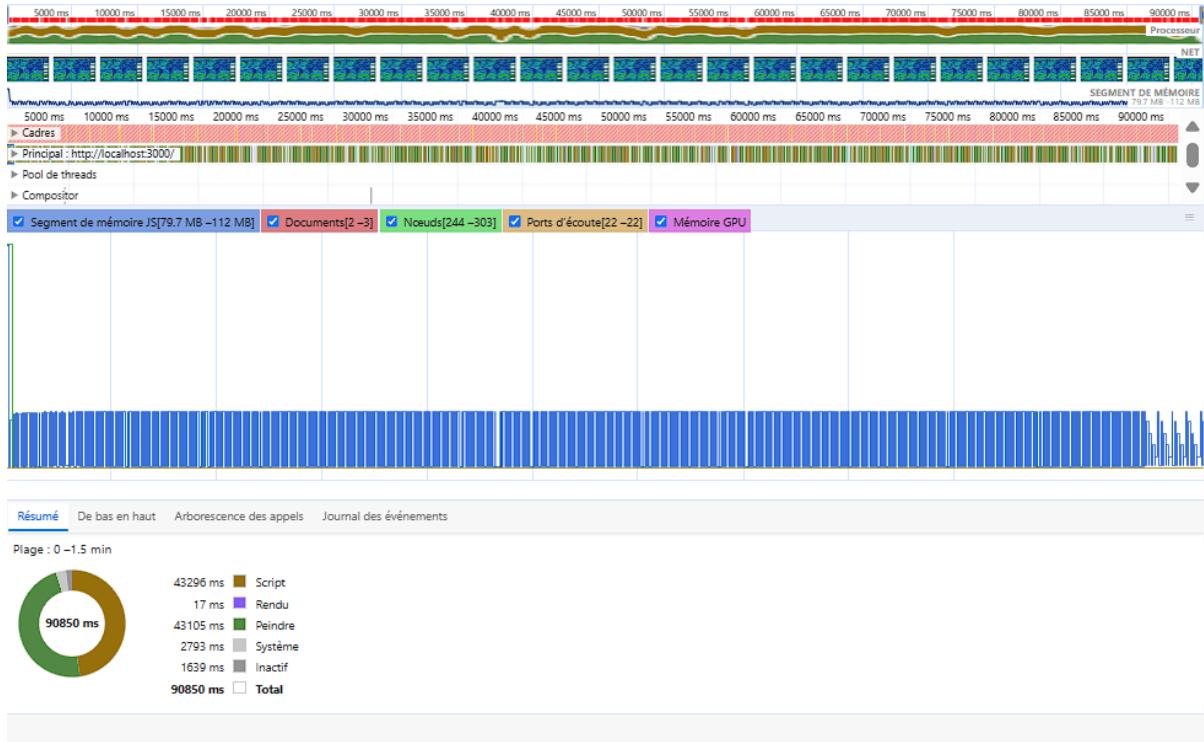


FIGURE 19 – Résultat des performances avec 70 000 particules

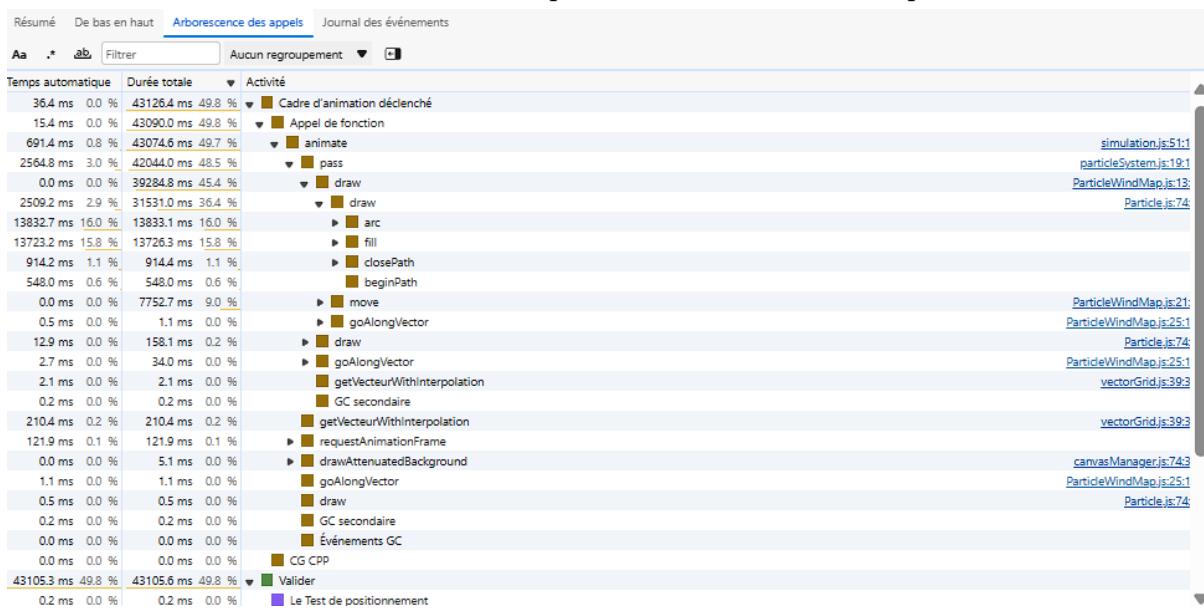


FIGURE 20 – Arborescence des appels de fonctions

On observe que l'on est grandement limité par nos fonctions de dessins des particules. Les parties en vert, étiquetées "peindre" ou "valider", représentent généralement les périodes où le navigateur est en train de calculer des styles ou de faire des calculs de mise en page.

5 Gestion du Projet

Dans la gestion de notre projet de système de particules, nous avons adopté une approche agile, en utilisant des sprints de deux semaines pour assurer un développement continu et une amélioration régulière des fonctionnalités. Cette partie détaille les pratiques de versioning, la planification des sprints et la collaboration d'équipe.

5.1 Organisation

Avec notre encadrante, nous nous sommes mis d'accord pour organiser le développement en sprints de deux semaines, où chaque sprint commençait par une réunion de revue de sprint et une rétrospective puis sur la planification des tâches à faire pour les deux prochaines semaines. Cela a permis d'avoir des retours pendant le développement et d'avoir des objectifs à court terme précis.

5.2 Versioning, Utilisation de GitHub

Pour le contrôle de version et la collaboration, nous avons utilisé GitHub, outil qui a facilité la gestion de notre code source et la collaboration entre les membres du groupe. Le dépôt GitHub de notre projet est accessible publiquement et peut être consulté via le lien suivant : <https://github.com/JMarguery/ParticleSystemPP2GroupH>. Cela permet à d'autres personnes si elles le souhaitent de savoir plus de détails sur notre implémentation.

5.3 Changements majeurs

Un des changements majeurs a été la modification de la façon de dessiner la trace que laisser le passage des particules, cela a nécessité de mettre à jour le code de chaque système à particules. Ce changement était nécessaire, car notre ancienne méthode, bien que permettant de faire des effets de traces plus complexes visuellement, ne permettait pas d'atteindre un nombre suffisamment élevé de particules pour avoir une densité suffisante, notamment sur la carte des vents.

6 Bilan et Conclusions

6.1 Perspectives d'amélioration

Pour les perspectives d'amélioration du projet, nous envisageons quelques améliorations qui pourraient significativement améliorer notre projet :

- Convertir le code pour passer à une API comme WebGL ou WebGPU pour pouvoir faire des calculs via le GPU, qui permettrait d'avoir beaucoup plus de particules à l'écran tout en ayant de meilleures performances.
- Ajout de paramètres supplémentaires à modifier pour les utilisateurs comme permettre à l'utilisateur de téléverser son propre fichier pour la carte des vents ou la carte des élections.

6.2 Conclusion

À travers les différents systèmes développés, nous avons exploré plusieurs aspects des systèmes à particules, depuis les interactions physiques comme la détection des collisions basiques jusqu'aux représentations plus abstraites comme celles des données électorales ou des données météorologiques. Chaque système a présenté ses propres défis comme la gestion de la performance ou la mise en forme et l'interprétation des données en entrée.

En plus de ces défis, la visualisation de données en était un autre, nos systèmes affiant des données spatio-temporelles doivent permettre aux utilisateurs de percevoir des tendances, de détecter les anomalies, et de comprendre des événements complexes à travers des représentations intuitives.

Références

- [1] Vladimir Agafonkin. How I built a wind map with WebGL. pages 203–220, 2017.
- [2] Cameron Beccario. earth : : a global map of wind, weather and ocean conditions, 2024.
- [3] Ministère de l'Intérieur et des Outre-Mer. Jeux de données - data.gouv.fr, 2022.
- [4] National Oceanic and Atmospheric Administration (NOAA). NOMADS at ncep.noaa.gov, 2024.
- [5] William T Reeves. Particle systems—a technique for modeling a class of fuzzy objects. In *Seminal graphics : pioneering efforts that shaped the field*, pages 203–220. 1998.
- [6] Wikipedia. Election présidentielle en France - Wikipedia, 2024.

7 Annexe

```
static getVecteurWithInterpolation(coordX, coordY) {
    const centerX = this.cols / 2;

    const coordXWithTransformations = (coordX - CanvasManager.offsetX) /
        CanvasManager.zoomScale;
    const coordYWithTransformations = (coordY - CanvasManager.offsetY) /
        CanvasManager.zoomScale;

    let x = ((coordXWithTransformations/ CanvasManager.canvas.width) *
        this.cols + centerX) % this.cols;
    let y = (coordYWithTransformations / CanvasManager.canvas.height) *
        this.rows;

    x = (x + this.cols) % this.cols;
    y = Math.max(0, Math.min(this.rows - 1, y));

    const x0 = Math.floor(x);
    const x1 = (x0 + 1) % this.cols;
    const y0 = Math.floor(y);
    const y1 = Math.min(y0 + 1, this.rows - 1);

    const vectorBottomLeft = this.vecteurs[this.index(x0, y0)];
    const vectorBottomRight = this.vecteurs[this.index(x1, y0)];
    const vectorTopLeft = this.vecteurs[this.index(x0, y1)];
    const vectorTopRight = this.vecteurs[this.index(x1, y1)];

    const fx1 = x - x0;
    const fx0 = 1 - fx1;
    const fy1 = y - y0;
    const fy0 = 1 - fy1;
    const interpolatedX = vectorBottomLeft.x * fx0 * fy0 +
        vectorBottomRight.x * fx1 * fy0 +
        vectorTopLeft.x * fx0 * fy1 + vectorTopRight.x * fx1 * fy1;
    const interpolatedY = vectorBottomLeft.y * fx0 * fy0 +
        vectorBottomRight.y * fx1 * fy0 +
        vectorTopLeft.y * fx0 * fy1 + vectorTopRight.y * fx1 * fy1;
    return { x: interpolatedX, y: interpolatedY};
}
```

FIGURE 21 – Code de l’interpolation bilinéaire