

Travail pratique

Lignes de champs électriques

El Kharroubi Michaël
(et un tout petit peu Orestis Malaspinas)

1 But

- Simuler un phénomène physique vu en cours à l'aide du langage C
- Visualiser les lignes de champs électriques générées par N particules

2 Rappel théorique

Comme nous l'avons vu en classe, chaque particule possédant une charge Q induit un champs E qui influe sur l'espace autour d'elle. Il est possible de représenter ce champs électrique à l'aide d'un champs de vecteur. Lorsque que l'on a une seule particule chargée, chaque vecteur décrit en un point l'action induite par la particule chargée à cette distance. L'intensité de ce champs à une distance r de la particule est donné par la formule suivante :

$$E = k \frac{Q}{r^2} \quad \text{avec } k = \frac{1}{4\pi\epsilon_0} \quad (1)$$

En ce qui concerne sa direction, si $Q > 0$ alors le vecteur va chercher à s'éloigner de la particule, et si $Q < 0$ alors le vecteur est dirigé vers la particule.

En réalité, on rencontre souvent (très souvent) plus d'une particule chargée, dans ce cas, on se sert du principe de superposition des champs électriques. Le vecteur en un point représente alors la résultante des actions induites par chacune de nos particules chargées.

Une autre représentation du champs électrique peut se faire à l'aide de courbes appelées lignes de champs.

Pour réaliser cette simulation numérique, nous devons commencer par définir notre univers discret. Dans le cadre de ce travail, notre univers est un rectangle $[0; 1] \times [0; 1]$.

Chaque particule possède une position appartenant à notre univers, ainsi qu'une charge (multiple de la charge élémentaire).

Pour dessiner numériquement une courbe, une méthode consiste à l'approximer à l'aide d'un ensemble de segments de droites reliant des points appartenant à

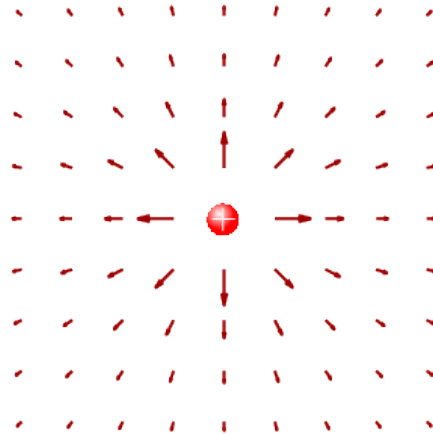


Figure 1: Champs de vecteurs représentant de champs électrique d'une charge positive. Source: Wikipédia, <https://bit.ly/3kUEcGu>.

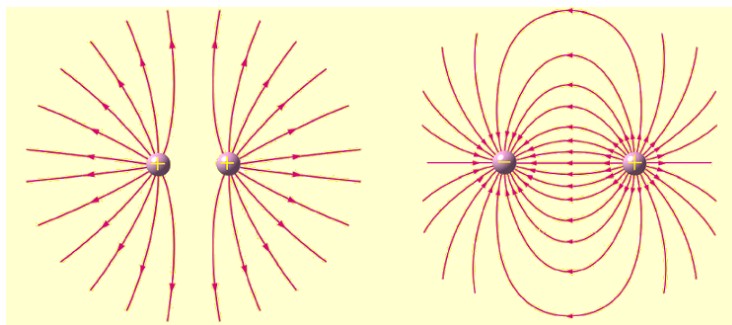


Figure 2: Lignes de champs électrique en présence de deux charges positives (gauche) et deux charges opposées (droite). Les lignes de champs sont sortantes de la charge positive, et entrantes dans la charge négative. Source: Wikipédia, <https://bit.ly/3dPsUk2>.

cette courbe. Pour dessiner notre ligne de champs, nous devons donc trouver un ensemble de points appartenant à cette ligne. Pour cela on travaille ainsi :

1. On tire aléatoirement un point dans notre univers P_0 .
2. On assigne $P = P_0$
3. On calcule $P_{suivant}$ à partir de l'intensité et la direction du champs en P , ce qui nous donne $P_{suivant} = P + \sum_i^N E_i \cdot \frac{q_i \vec{P}}{\|q_i \vec{P}\|}$ avec $q_i \vec{P}$ le vecteur allant de la particule chargée q_i au point P . Dans notre cas, on souhaite avancer d'une même distance à chaque fois, et qui ne soit pas trop grande. On va donc commencer par normaliser le champs, puis on va ajouter une constante δx qui est fixe pour notre programme et qui est définie en fonction de la taille de notre fenêtre. Ce qui nous donne $P_{suivant} = P + \delta x \cdot \frac{\vec{E}}{\|\vec{E}\|}$ avec $\delta x = \frac{1}{\sqrt{\text{largeur}^2 + \text{hauteur}^2}}$ et $\vec{E} = \sum_i^N E_i \cdot \frac{q_i \vec{P}}{\|q_i \vec{P}\|}$.
4. Si P et $P_{suivant}$ sont à une distance des particules, supérieure à un seuil choisi et qu'ils sont toujours dans notre univers, on trace un segment entre P et $P_{suivant}$. Sinon, on quitte notre boucle.
5. On assigne $P = P_{suivant}$ et on revient à l'étape 3

Cette méthode nous permet de dessiner partiellement notre ligne de champs. Comme nous partons d'un point aléatoire, il faut également dessiner le reste de la ligne en allant dans le sens opposé. Il faudra donc réitérer notre processus, en partant à nouveau de la deuxième étape, mais en calculant $P_{suivant} = P - \delta x \cdot \frac{\vec{E}}{\|\vec{E}\|}$.

3 Énoncé

Vous allez développer une simulation de lignes de champs générée par un ensemble de particules en C, et visualiser le résultat à l'aide de la librairie SDL. Vous devez réutiliser la librairie de vecteurs en deux dimensions réalisée au premier semestre. Deux fichiers `utils.h` et `utils.c` vous seront fournis avec l'énoncé. Le fichier `utils.c` contient des méthodes afin de vous faciliter la réalisation de ce TP. Pour récupérer ces fichiers vous pouvez exécuter la commande

```
wget https://malaspinas.academy/phys/field_lines/utils.tar.gz
```

Ce travail va se diviser en deux parties.

3.1 Dessin

Pour pouvoir représenter ce que vous allez calculer dans la deuxième partie, vous allez devoir dessiner des droites et des cercles à l'aide des méthodes de Bresenham et d'Andres (voir les liens ci-dessous). Pour cela vous implémenterez les fonctions suivantes :

```
typedef struct
{
    uint32_t row;
    uint32_t column;
} coordinates_t;
```

```
void gfx_draw_line(struct gfx_context_t *ctxt, coordinates_t p0,
                  coordinates_t p1, uint32_t color);
```

```
void gfx_draw_circle(struct gfx_context_t *ctxt, coordinates_t c,
                    uint32_t r, uint32_t color);
```

Pour ce faire vous devez implémenter les algorithmes de dessin du [segment de droite de Bresenham](#) et du [cercle d'Andres](#).

Pour tester votre fonction de dessin de droites, vous dessinerez dans une fenêtre de 100×100 les droites suivantes :

- $(50, 50) \rightarrow (75, 50)$ ¹, $(50, 50) \rightarrow (72, 62)$, $(50, 50) \rightarrow (62, 72)$;
- $(50, 50) \rightarrow (50, 75)$, $(50, 50) \rightarrow (38, 72)$, $(50, 50) \rightarrow (28, 62)$;
- $(50, 50) \rightarrow (25, 50)$, $(50, 50) \rightarrow (28, 38)$, $(50, 50) \rightarrow (37, 28)$;
- $(50, 50) \rightarrow (50, 25)$, $(50, 50) \rightarrow (62, 28)$, $(50, 50) \rightarrow (72, 37)$.

3.2 Physique

Comme nous l'avons vu durant la partie théorique, vous allez devoir calculer différents points appartenant à différentes lignes de champs. Pour cela vous implémenterez les fonctions suivantes :

```
typedef struct
{
    double q;
    vec2 pos;
} charge_t;

// Compute E*qP/norm(qP)
// Return false if norm(qP) < eps
bool compute_e(charge_t c, vec2 p, double eps, vec2 *e);

// Compute the normalized sum of Ei*qiP/norm(qiP)
// Return false if for some qiP, norm(qiP) < eps
bool compute_total_normalized_e(charge_t *charges, int num_charges, vec2 p,
                                double eps, vec2 *e);

// Compute and then draw all the points belonging to a field line,
// starting from pos0.
// Returns false if pos0 is not a valid position
// (for example if pos0 is too close to a charge).
static bool draw_field_line(struct gfx_context_t *ctxt, charge_t *charges,
                           int num_charges, double dx, vec2 pos0, double x0,
                           double x1, double y0, double y1);

// Draw all the charges
// A circle with minus sign for negative charges
// A circle with a plus sign for positive charges
```

¹Le segment reliant le point (x_0, y_0) au point (x_1, y_1)

```
static void draw_charges(struct gfx_context_t *context, charge_t *charges,  
    int num_charges, double x0, double x1, double y0, double y1);
```