TP de physique - Système solaire

Agnon Kurteshi

Zabiullah Ahmadi

2021

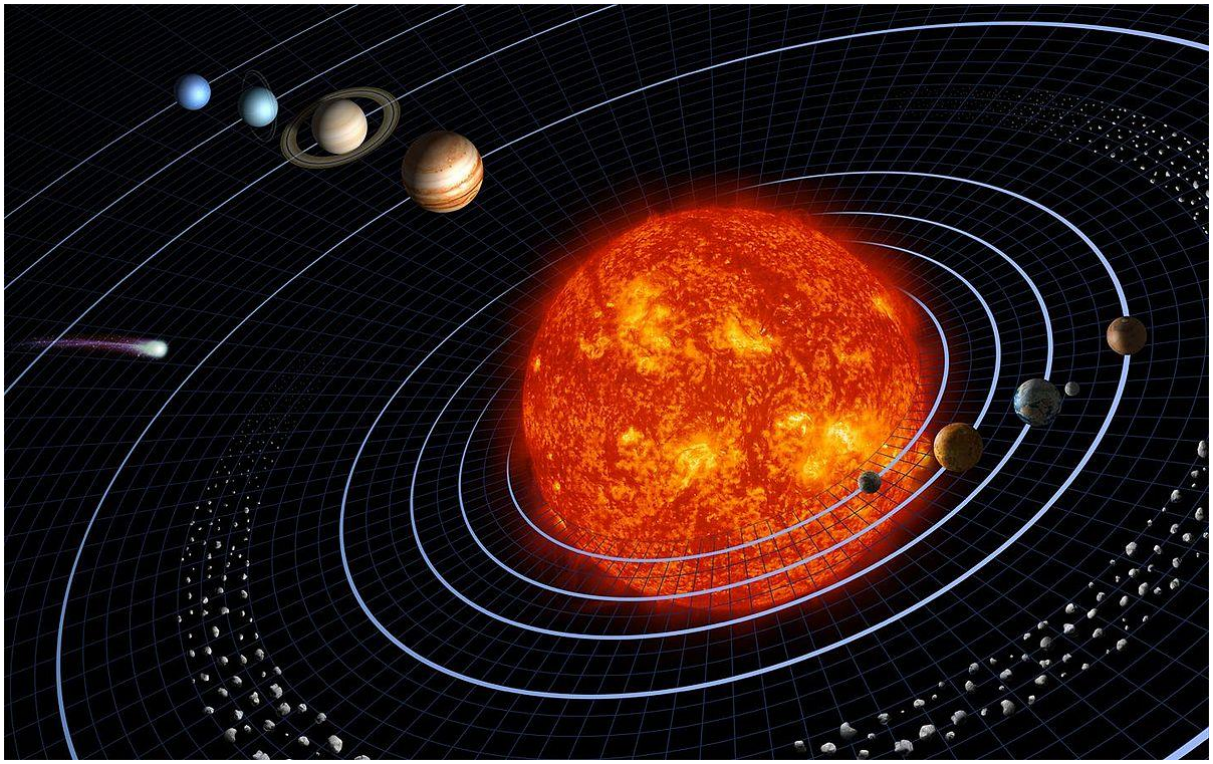


Table des matières

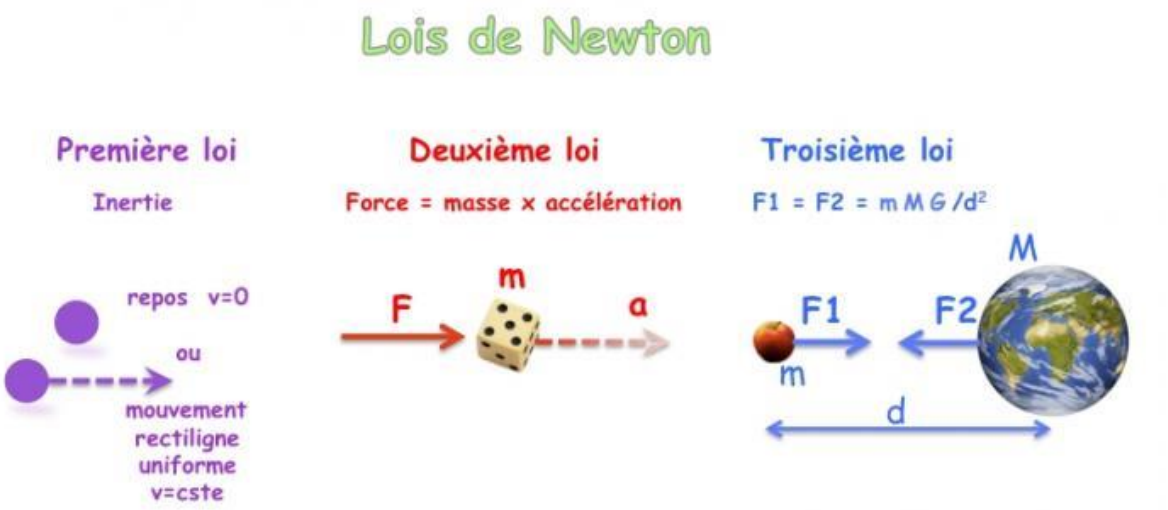
I. Présentation :

Pour ce travail pratique, il nous a été demandé de coder (en C), une simulation du système solaire. Pour ce faire, nous avons eu le droit à une Template déjà faite pour la partie graphique (donc, affichage des planètes, et fenêtre de l'application), et une libraire de vecteur (vec2.c) fait dans un précédent TP. Notre charge du travail a été de coder tout le reste, donc, l'initialisation du système solaire, le mouvement des planètes, calcul de force, excentricité, etc... Pour arriver à nos fins, nous avons dû utiliser les trois lois de Newton :

- 1ère loi de Newton : "Tout corps persévère dans l'état de repos ou de mouvement uniforme en ligne droite dans lequel il se trouve, à moins que quelque force n'agisse sur lui, et ne le contraigne à changer d'état"

- 2ème loi de Newton : "Les changements qui arrivent dans le mouvement sont proportionnels à la force motrice ; et se font dans la ligne droite dans laquelle cette force a été imprimée"

- 3ème loi de Newton : "L'action est toujours égale à la réaction ; c'est-à-dire que les actions de deux corps l'un sur l'autre sont toujours égales et de sens contraires"



II. Démarche du travail :

1. Présentation des fonctions

Pour faire ce TP, nous avons utilisé pas quelques fonctions indispensables pour que la simulation du système solaire fonctionne. Voici le fichier planet.h qui montre les fonctions :

```
planet_t create_planet(double mass,double rayon, double excentricity, double demi_grand_axe, uint32_t color);

system_t create_system(int32_t nb_planets);

vec2 intial_position(planet_t planet, double delta_t, system_t *system);

vec2 force_gravitationelle_planet(planet_t a, planet_t b);

vec2 planet_resultant_force(system_t *system, planet_t planet);

vec2 planet_acceleration(planet_t planet, system_t *system);

vec2 next_position(planet_t planet, double delta_t, system_t *system);

void show_system(struct gfx_context_t *ctxt, system_t *system);

void update_system(system_t *system, double delta_t);
```

- La fonction create_planet nous permet de créer une planète tout en lui fournissant les données de la masse, du rayon, de l'excentricité, le périhélie et une couleur
- La fonction create_system prend en paramètre le nombre de planètes et permet d'initialiser l'étoile (Soleil) au centre de l'écran et de nous créer le système évidemment
- La fonction initial_position permet de donner une vitesse à la planète pour pas qu'elle fonce sur le soleil
- La fonction force_gravitationelle_planet permet de calculer la force de gravité entre deux planètes
- La fonction planet_acceleration calcul l'accélération en utilisant la force et la masse
- La fonction next_position nous trouve la position de la planète après avoir ajouté delta_t au temps actuel
- La fonction show_system permet d'afficher l'étoile et les planètes sur l'écran
- La fonction update_system est une des fonctions les plus utiles (le reste l'est aussi), car c'est grâce à elle que la planète va pouvoir bouger

Il y a d'autres fonctions dans notre code, mais les fonctions ci-dessus sont les pièces majeures du fonctionnement de notre simulation.

2. Initialisation du système solaire

Pour commencer, nous avons dû placer toutes les planètes dans une matrice carrée allant de –1 à 1, où le 0 est le centre de l'écran. Pour ce faire, nous avons utilisé la librairie vec2.c, où il y avait une fonction qui permettait de calculer la position du vecteur sur un écran. Grâce à ça, nous avons pu, avec la libraire GFX, afficher les planètes sur notre écran.

Nous avons eu quelques soucis lors de la mise en place du système solaire. Comme nous utilisions des valeurs réels (qui se comptes en millions et milliards) pour la taille, distance et masse, il fallait trouver un coefficient de proportion, entre la taille des planètes. L'étoile quant à elle, n'est pas du tout proportionnelle, car il est impossible de représenter la vraie proportion du soleil sur un petit écran.

Donc, on a initialisé l'étoile et les planètes, et chacune des planètes initialisées possède sa propre masse, sa taille, son rayon, etc... Ces valeurs nous sont utiles pour tous les calculs permettant la simulation.

Pour la partie système, il n'y a que l'étoile qui reçoit des données, et elle reçoit comme position un vecteur nul pour le placer au centre de l'écran.

```
system_t create_system(int nb_planets) {

  system_t system;
  system.nb_planets = nb_planets;
  system.star = create_planet(M_SOLEIL,10,0,0, COLOR_YELLOW);
  system.star.pos = vec2_create_zero();
  return system;

}
```

Figure 1: code de création du système

3. Initialisation des planètes

Maintenant que nous avons le système solaire, il faut bien sûr lui ajouter des planètes qui graviteront autour de l'étoile. Pour ce faire, nous créeons une planète grâce à la fonction create_planet ci-dessous.

```
planet_t create_planet(double mass,double rayon, double excentricity, double demi_grand_axe, uint32_t color){

  planet_t planet;
  planet.mass = mass;
  planet.rayon = rayon;
  planet.pos = vec2_create(demi_grand_axe * (1 - excentricity), 0);
  planet.prec_pos =  vec2_create_zero();
  planet.planet_excentricity = excentricity;
  planet.planet_demi_grand_ax = demi_grand_axe;
  planet.color = color;

  return planet;
```

Figure 2: code de création de la planète

Cette fonction nous permet d'initialiser la planète avec une masse, un rayon, son excentricité et son périhélie (valeurs réeles). Ensuite, on crée un vecteur pour la position afin de pouvoir l'afficher sur l'écran. Pour le calcul de ce point, on utilise le périhélie et l'excentricité. On a décidé de placer toutes les planètes à leur périhélie, c'est pour ça qu'on utilise le périhélie comme point de départ lors de la création de la planète. Comme au lancement de la simulation du système solaire il n'y a pas de position précédente, on lui attribue un vecteur nul.

4. Affichage des planètes et du soleil

Jusqu'à présent, nous avons juste créé les planètes et le soleil, mais n'apparaît à l'écran. C'est normal, pour les faire apparaître, il faut utiliser la fonction show_system créé par nos soins. La voici ci-dessous :

```
void show_system(struct gfx_context_t *ctxt, system_t *system) {

  // affiche le soleil
  coordinates sun_cord = vec2_to_coordinates(system->star.pos,SCREEN_WIDTH,SCREEN_HEIGHT);
  draw_full_circle(ctxt,sun_cord.column, sun_cord.row, system->star.rayon, system->star.color);

  // affiche les planètes
  for (uint32_t i = 0; i < system->nb_planets; i++){

    // calcule la coef de proportion
    long coef = system->planets[system->nb_planets-1].planet_demi_grand_ax;
    double scalar = 1.0/coef;

    // coord de centre de la planète
    vec2 center = vec2_mul(scalar, system->planets[i].pos);
    // rayon de la planète
    double rayon = (system->planets[i].rayon /coef  * 100000) * 5;

    coordinates screen_pos = vec2_to_coordinates(center, SCREEN_WIDTH, SCREEN_HEIGHT);

    draw_full_circle(ctxt, screen_pos.column, screen_pos.row, rayon,system->planets[i].color);
```

Figure 3: code permettant d'afficher le système

On sait que la position du soleil et des planètes sont exprimés sous formes de vecteur. Or, nous ne pouvons pas utiliser un vecteur pour afficher ces derniers. En effet, nous savons que l'écran est une matrice de [-1, 1], et donc, la seule manière de pouvoir afficher les vecteurs, c'est d'utiliser la fonction vec2_to_coordinates qui permet de transformer un vecteur en axe X et Y. Ensuite, on utilise la fonction déjà donnée pour dessiner une planète (draw_full_circle).

5. Fonctions liées à la physique et mouvement des planètes

Afin de faire bouger les planètes, nous allons devoir utiliser toutes les formules de physiques utiles, donc, la force résultante, l'accélération des planètes, la force gravitationnelle, etc… On peut bien le comprendre qu'on a fait une fonction pour la plupart de ces formules afin de simplifier le travail.

- La force gravitationnelle (fonction force_gravitationelle_planet) :

```
vec2 force_gravitationelle_planet(planet_t a, planet_t b) {
  vec2 r_a_b = vec2_sub(b.pos, a.pos);
  vec2 m_f = vec2_mul(G * ((a.mass * b.mass) / pow(vec2_norm(r_a_b), 3)), r_a_b);

  return m_f;
}
```

Figure 4: code pour la force gravitationnelle

C'est une fonction toute simple. On soustraire la position de la planète A avec celle de la planète B (en termes de vecteur). Cela nous donne la distance entre la planète A et B.

Ensuite, on suit la formule :

$$\vec{F}_{BA} = G\frac{m_A m_B}{\|\vec{r}_{AB}\|^3}\vec{r}_{AB},$$

- La force résultante (fonction planet_resultant_force) :

```c
vec2 planet_resultant_force(system_t *system, planet_t planet) {

  vec2 resulting_force = vec2_create_zero();

  //la force gravitationelle entre soleil et la planète
  vec2 force_g = force_gravitationelle_planet(planet, system->star);
  resulting_force = vec2_add(resulting_force, force_g);

  // force resultantes des planètes sur l'un l'autre
  for (uint8_t i = 0; i < system->nb_planets; i++) {

    // annule la force resultante sur elle-meme
    if (planet.mass == system->planets[i].mass)
        continue;

    vec2 f_g = force_gravitationelle_planet(planet, system->planets[i]);
    resulting_force = vec2_add(resulting_force, f_g);
  }

  return resulting_force;
}
```

Figure 5: code de la force résultante d'une planète

Cette fonction permet de calculer la force résultante d'une planète qu'exerce toutes les planètes et le soleil sur elle. Dans un premier temps, on calcule la force du soleil avec la planète, ensuite, on calcule la force des autres planètes. Pour finir, on fait une condition qui exclut la force de la planète elle-même, car elle ne peut pas exercer une force sur elle-même.

- L'accélération des planètes (fonction planet_acceleration) :

```c
vec2 planet_acceleration(planet_t planet, system_t *system) {

  vec2 force_resultant = planet_resultant_force(system, planet);
  //a = 1/m * f
  return vec2_mul(1 / planet.mass, force_resultant);
}
```

Figure 6: code pour l'accélération d'une planète

Avec la fonction de la force résultante, on récupère la force résultante exercée sur une planète, et on calcule la vitesse grâce à la deuxième loi de Newton (Accélération = Masse x force résultante).

- La vitesse d'une planète (fonction calcul_de_vp) :

```c
vec2 calcul_de_vp(planet_t planet, system_t *system) {

  vec2 vector_normal = vec2_create(-planet.pos.y, planet.pos.x);
  double scalar = sqrt((G * system->star.mass * (1 + planet.planet_excentricity)) /
  (planet.planet_demi_grand_ax * (1 - planet.planet_excentricity)));
  vec2 v_unitaire_vp = vec2_normalize(vector_normal);
  vec2 vp = vec2_mul(scalar, v_unitaire_vp);
  return vp;
}
```

Figure 7: calcul de la vitesse d'une planète

Pour cette fonction, on va utiliser la fonction suivante :

$$\vec{v}_p(0) = \sqrt{\frac{GM_\odot(1+e_p)}{a_p(1-e_p)}} \cdot \frac{\vec{r}_{p\perp}}{\|\vec{r}_{p\perp}\|},$$

On va commencer par créer le vecteur perpendiculaire. Il va nous servir à faire une trajectoire circulaire pour nos planètes. Pour ce faire, on prend la position « x » et « y » de la planète, et on modifie le « x » en « -y » et le « y » en « x ». La première partie de la formule (la racine), on va la mettre dans une variable « scalar » afin de pouvoir la multiplier à un vecteur (grâce à la librairie vec2). Ensuite, la deuxième partie, c'est la normalisation du vecteur perpendiculaire qui va donner la direction de la planète (vecteur unitaire). Maintenant, on a le vecteur unitaire et le scalaire, on les multiplie entre eux, et voilà maintenant on a la vitesse de la planète.

- Mouvement de la planète (next_position)

```
vec2 next_position(planet_t planet, double delta_t, system_t *system){

  vec2 accelleration = planet_acceleration(planet, system);

  //xp (t + delta_t)
  vec2 next_pos = vec2_add(vec2_sub(vec2_mul(2, planet.pos), planet.prec_pos), vec2_mul(pow(delta_t, 2), accelleration));

  return next_pos;

}
```

Figure 8: code calculant la prochaine position

Avec la fonction planet_acceleration, on va récupérer l'accélération de la planète pour pouvoir l'utiliser dans la formule de l'évolution du système.

Maintenant qu'on a l'accélération, il suffit de prendre la formule suivante et de l'appliquer :

$$\vec{x}_p(t+\Delta t) = \vec{x}_p(t) + \Delta t\vec{v}_p(t) + \frac{(\Delta t)^2}{2}\vec{a}_p(t)$$

Cette formule permet avec delta_t de faire évoluer la planète dans le temps, delta_t étant exprimé en secondes. Plus delta_t est grand plus le système solaire va être rapide.

- Mise à jour du déplacement

```
void update_system(system_t *system, double delta_t) {

  for (uint32_t i = 0; i < system->nb_planets; i++) {

    vec2 prec_pos = system->planets[i].pos;
    system->planets[i].pos = next_position(system->planets[i], delta_t, system);
    system->planets[i].prec_pos = prec_pos;
  }

}
```

Figure 9: code de la mise à jour des positions des planètes

Cette fonction va mettre à jour la position de toutes les planètes. Si nous n'avions pas cette fonction, tout le code vu précédemment ne ferait jamais bouger les planètes. Tout ce qu'on fait avec cette fonction, c'est de stocker la position actuelle dans la position précédente (besoin pour nos calculs) et recalculer la prochaine position de la planète.

III. Table des illustrations