

Rapport du travail pratique "Simulation d'un système planétaire"

Ce travail pratique sur la "Simulation d'un système planétaire sur un plan à l'aide des lois de Newton" est réalisé par M. Gawen ACKERMANN et M. Florian BURGNER, dans le cadre du cours de Physique appliquée à l'ingénierie 1 (ISC_123).

But

Le but de ce travail est de créer une simulation d'un système planétaire, il nous permet notamment d'appliquer les 3 lois de Newton vu dans le cours de Physique ainsi que de manipuler des vecteurs. La première partie de ce travail consiste à avoir 4 planètes : Mercure, Vénus, Terre et Mars ; en orbite autour du Soleil (en fonction des vraies valeurs et des lois physiques bien sûr). La deuxième partie consiste à ajouter plusieurs autres planètes avec des valeurs qui n'existent pas.

Installation

Nous avons pris la décision de changer de librairie graphique (en accord avec l'enseignant) afin d'avoir de meilleures performances de rendu. La librairie que nous avons choisi d'utiliser s'appelle FreeGLUT et est basée sur OpenGL.

Pour pouvoir compiler le projet, il faut donc impérativement installer la librairie FreeGLUT avec la commande suivante :

```
sudo apt-get install freeglut3-dev
```

Pour compiler le projet, il faut premièrement se rendre dans le dossier du code source avec la commande ci-dessous :

```
cd src
```

puis exécuter la commande make afin de compiler automatiquement le projet :

```
make
```

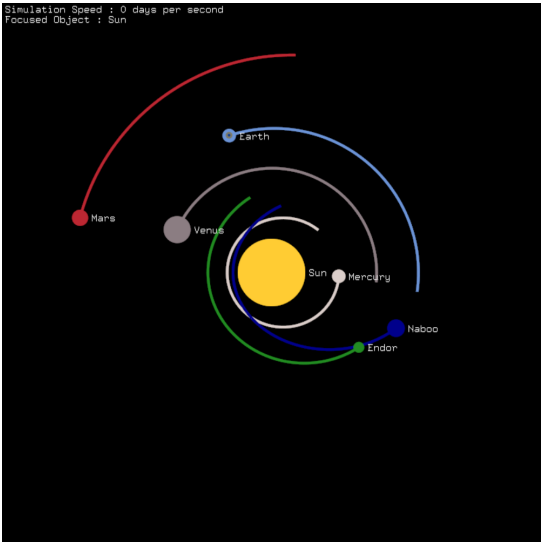
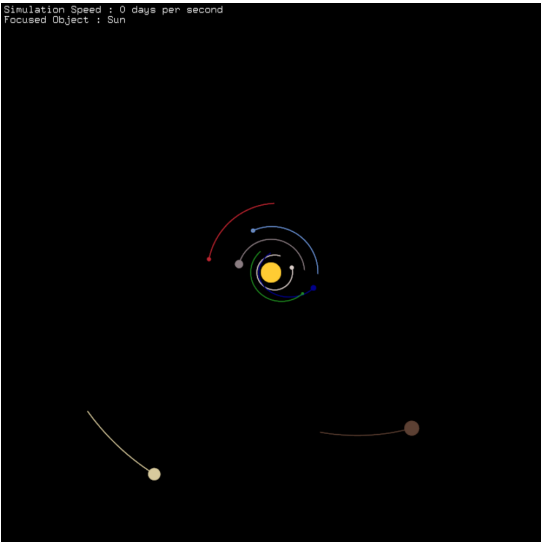
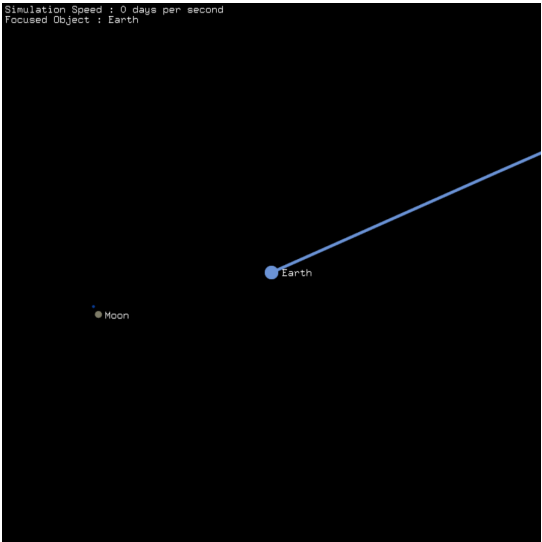
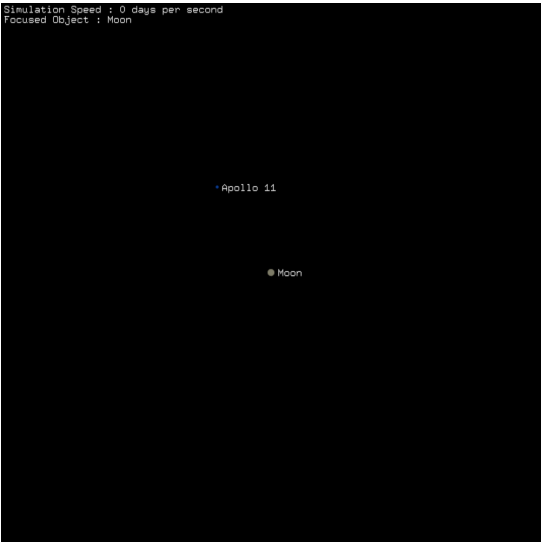
Pour lancer le projet, exécuter la commande suivante :

```
./planetary_system
```

Le résultat du projet en vidéo

Nous vous conseillons vivement de regarder la vidéo (en annexe de ce rapport) que nous avons créée pour avoir une idée plus claire du résultat du projet.

Le résultat du projet en images

<div><p>Affichage par défaut</p><p>Simulation Speed : 0 days per second Focused Object : Sun</p><p>The screenshot shows a top-down view of a simulated solar system. At the center is a large yellow circle labeled 'Sun'. Several planets are shown in orbit around it, each with a colored trail indicating its path: Mars (red), Earth (blue), Venus (grey), Mercury (white), Naboo (dark blue), and Endor (green). The orbits are elliptical and intersect at various points. The background is black.</p></div>	<div><p>Affichage dézoomé</p><p>Simulation Speed : 0 days per second Focused Object : Sun</p><p>This screenshot shows the same solar system simulation but zoomed out significantly. The Sun is now a small yellow dot in the center. The orbits of the planets are visible as thin, curved lines. The background is black.</p></div>
<div><p>Système Terre-Lune</p><p>Simulation Speed : 0 days per second Focused Object : Earth</p><p>The screenshot shows a close-up view of the Earth-Moon system. A large blue circle labeled 'Earth' is in the center. A smaller grey circle labeled 'Moon' is in orbit around it. The background is black.</p></div>	<div><p>Système Lune-Apollo 11</p><p>Simulation Speed : 0 days per second Focused Object : Moon</p><p>This screenshot shows a close-up view of the Moon-Apollo 11 system. A large grey circle labeled 'Moon' is in the center. A smaller brown circle labeled 'Apollo 11' is in orbit around it. The background is black.</p></div>

Explications détaillées du projet

Données du Soleil et des planètes Mercure, Venus, Terre et Mars

Comme demandé dans l'énoncé, nous avons effectué des recherches pour obtenir les données réelles du Soleil et des planètes Mercure, Venus, Terre et Mars. Nous avons listé ces données dans le tableau ci-dessous :

Nom de l'objet céleste	Masse (en kg)	Excentricité	Demi-grand axe (en m)
Sun	$1988500 * 10^{24}$	/	/
Mercury	$0.33010 * 10^{24}$	0.2056	$57.909 * 10^9$
Venus	$4.8673 * 10^{24}$	0.0067	$108.209 * 10^9$
Earth	$5.9722 * 10^{24}$	0.0167	$149.596 * 10^9$
Mars	$0.64169 * 10^{24}$	0.0935	$227.923 * 10^9$

Toutes nos données viennent du service fournis par la NASA nommée "NASA Space Science Data Coordinated Archive".

- <https://nssdc.gsfc.nasa.gov/planetary/factsheet/>
- <https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html>
- <https://nssdc.gsfc.nasa.gov/planetary/factsheet/mercuryfact.html>
- <https://nssdc.gsfc.nasa.gov/planetary/factsheet/venusfact.html>
- <https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html>
- <https://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.html>

Nous avons aussi ajouté 1 autre planète réelle (Jupiter), 2 satellites (la Lune orbitant autour de la Terre et Apollo 11 orbitant autour de la Lune, pour les observer, il faut zoomer suffisamment tout en suivant la Terre ou la Lune) et 3 fausses planètes (Naboo, Endor, Coruscant). Au total, il y a 1 étoile, 8 planètes et 2 satellites.

Fonctionnement de la simulation

Calcul de la nouvelle position et de l'accélération appliquée à chaque Δt sur chaque objet céleste

Initialisation

1. Pour chaque objet céleste à créer ;
 1. Création de l'objet céleste en appelant la fonction `celestial_object_create` ;
 1. (dans la fonction `celestial_object_create`) ;
 2. Calcul du périapside en fonction du demi-grand axe et de l'excentricité.
1. Pour chaque objet céleste (excepté le Soleil) ;
 1. Calcul de la première position en appelant la fonction `celestial_object_first_update` ;

1. (dans la fonction `celestial_object_first_update`) ;
 2. Calcul de la vitesse au périapside ;
 3. Calcul de la vélocité en multipliant la vitesse à la norme du vecteur perpendiculaire à la distance entre l'objet céleste et le Soleil ;
 4. Calcul de l'accélération subit par l'objet céleste ;
 5. Calcul de la nouvelle position en fonction de la vitesse au périapside et l'accélération subit par l'objet céleste ;
1. Notez que le Δt est de 1 seconde pour l'initialisation.

À chaque mise à jour

1. Pour chaque objet céleste (excepté le Soleil) ;
 1. Calcul de la position en appelant la fonction `celestial_object_update` ;
 1. (dans la fonction `celestial_object_update`) ;
 2. Calcul de la variation du Δt (`interval / previous_interval`) ;
 1. Notez que le `previous_interval` correspond au Δt précédent tandis que la variable `interval` correspond au Δt actuel, à l'initialisation la variable `previous_interval` vaut 1 seconde, si le prochain Δt est de 100 secondes alors la "vitesse" sera multiplié par $100 / \text{previous_interval}$ pour garder la cohérence du MRU ;
 3. Calcul de la nouvelle position en fonction de la position, de la position précédente et de la variation du Δt (**1e loi de Newton**) ;
 4. Calcul de l'accélération subit par l'objet céleste ;
 5. Conversion de l'accélération en distance en fonction du Δt et ajout de cette distance à la nouvelle position.

Calcul de l'accélération subit par un objet céleste

Par la **3e loi de Newton** $\vec{F}_{BA} = -\vec{F}_{AB}$, autrement dit une force subit sur un corps A par un corps B est subit en réaction sur le corps B avec la même intensité et dans le sens opposé.

Nous avons utilisé la loi universelle de la gravitation et la **2e loi de Newton** pour calculer l'accélération subit par un objet céleste (noté *objet₂*).

2e loi de Newton : $\vec{F}_{Newton} = m \cdot \vec{a}$

Loi universelle de la gravitation : $\vec{F}_{Gravité} = G \frac{m_1 m_2}{\|\vec{r}\|^2} \hat{r}$

Soit un *corps₁* et un *corps₂*, l'accélération subit par le *corps₂* est :

$$\vec{F}_{Newton} = \vec{F}_{Gravité} \iff m_2 \cdot \vec{a}_2 = G \frac{m_1 m_2}{\|\vec{r}\|^2} \hat{r} \iff \vec{a}_2 = G \frac{m_1}{\|\vec{r}\|^2} \hat{r}$$

L'algorithme ci-dessous calcul la somme de toutes les \vec{a}_i subit par l'*objet₂*.

1. Pour chaque objet céleste, noté *objet₁* (excepté l'*objet₂*) ;
 1. Calcul de la distance \vec{r} entre l'*objet₁* et l'*objet₂* ;
 1. d = distance ;
 2. $\vec{r} = \vec{d}_{objet_1} - \vec{d}_{objet_2}$;
 2. Calcul de \vec{a}_i ;
 1. $\vec{a}_i = \frac{G \cdot m_1}{\|\vec{r}\|^2} \hat{r}$;
 3. On ajoute \vec{a}_i à \vec{a} subit par l'*objet₂*.

OpenGL

Afficher un cercle avec OpenGL

Puisque nous avons changé de librairie graphique, nous avons dû recréer une fonction permettant de dessiner un disque. Pour dessiner le disque, on calcule tous les points sur 360° (donc 360 points) par rapport à un rayon. Ensuite, OpenGL s'occupe de remplir le polygone avec la bonne couleur.

Fonction draw_disc

```
void draw_disc(Vector2 position, int32_t radius) {
    glBegin(GL_POLYGON);

    for (int32_t i = 0; i < 360; i += 1) {
        double theta = i * 3.14159265 / 180;
        double x = position.x + radius * cos(theta);
        double y = position.y + radius * sin(theta);
        glVertex2f(x, y);
    }

    glEnd();
}
```

La couleur à appliquer est calculée par rapport à sa valeur hexadécimale :

```
int32_t color = 0xFFFFFF;
int32_t r = (color & 0xFF0000) >> 16;
int32_t g = (color & 0x00FF00) >> 8;
int32_t b = (color & 0x0000FF) >> 0;
glColor3ub(r, g, b);
```

Fonctionnalités de l'interface graphique

Suivre la trajectoire d'une planète ou d'un satellite

Pour suivre la trajectoire d'une planète ou d'un satellite, il faut utiliser les **flèches directionnelles horizontales** du clavier. En haut à droite de l'écran, est indiqué quel objet est actuellement suivi.

Accélérer la vitesse de la simulation

Pour accélérer la vitesse de simulation, il faut utiliser la **flèche du haut** du clavier. La vitesse de simulation maximale est de 500 jours par seconde.

Décélérer la vitesse de la simulation

Pour décélérer la vitesse de simulation, il faut utiliser la **flèche du bas** du clavier. La vitesse de simulation minimale est de 0 jour par seconde.

Zoomer

Pour zoomer dans le système planétaire, il faut utiliser le roulement haut de la souris.

Dézoomer

Pour dézoomer dans le système planétaire, il faut utiliser le roulement bas de la souris.

Afficher/cacher les noms des objets célestes

Pour afficher ou cacher le nom des objets célestes, il faut appuyer sur la touche T.

Raisons pour lesquelles nous pensons que notre simulation fonctionne correctement

Pour vérifier que notre simulation fonctionne correctement, nous avons laissé tourner la simulation durant 100 ans et avons récupéré le périhélie et aphélie des planètes Mercure, Venus, Terre et Mars et les avons comparés aux vrais périhélie et aphélie. Nous avons pu observer que nos valeurs étaient très proche de la réalité et avons donc conclu que notre simulation fonctionne correctement.