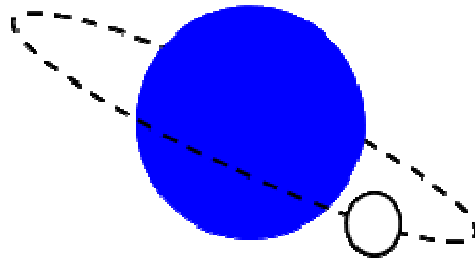
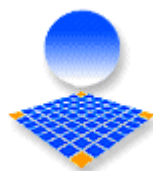


Rapport de projet informatique de l'Unité d'Enseignement ULIN405  
de la Licence informatique 2<sup>ème</sup> année effectué  
du 05 Mars au 14 Mai

# Modélisation, rendu et mouvement de la lune autour de la terre



Ísis DE SÁ ARAÚJO COSTA  
Hamza M'GUIL





## **Remerciements**

Nous voudrions remercier en premier notre tuteur Mme Mountaz Hascöet qui nous a orientés, guidés et expliqués toutes les démarches à suivre pour le bon déroulement de notre projet informatique.

Un grand merci à Bruno le petit ami d'Isis qui nous a été d'un grand soutien dans toute la partie conception du programme, il était aussi présent à toutes nos réunions à l'écoute de toutes nos contraintes rencontrées tout au long des deux derniers mois.

Nous tenons aussi à remercier le responsable de cette unité d'enseignement (projet informatique) Monsieur M. Meynard plus précisément, qui nous a donné cette chance de faire preuve de créativité, amélioré notre sens de travail en groupe et surtout sortir de ce monde des études théoriques et mettre en évidence nos compétences acquises pendant ces deux années de licence et de recherche en informatique, qui grâce à lui l'un de nous est présent aujourd'hui à l'Université Montpellier II. Je (Isis) tiens personnellement à lui présenter tous mes remerciements les plus sincères.



# Table des Matières

1.	Introduction .....	1
1.1	Généralités .....	1
1.2	Le sujet .....	1
1.3	Cahier des charges.....	2
2.	Organisation du projet .....	7
2.1	Organisation du travail.....	7
2.2	Choix des outils de développement .....	7
3.	Analyse du projet .....	9
4.	Développement.....	13
5.	Manuel d'utilisation .....	21
6.	Perspectives et conclusions .....	29
6.1	Perspectives.....	29
6.2	Conclusions.....	29



# **1. Introduction**

## **1.1 Généralités**

Ce projet informatique est une unité d'enseignement qui se décrit comme une analyse, conception et développement en groupe, d'un projet informatique dans le langage de programmation C. Ce programme doit être présenté lors d'une soutenance orale et accompagner de ce rapport.

Notre projet à nous consiste à concevoir un programme d'une vraie application qui modélise les mouvements de la lune autour de la terre en respectant les règles de la physique réelle.

En ce qui concerne le rapport, nous avons eu l'idée d'utiliser 'Google docs' pour pouvoir partager et modifier notre rapport de n'importe où et n'importe quand. Ce qui nous a permis de faire un travail en ligne et collaboratif en gagnant du temps.

## **1.2 Le sujet**

Le sujet de ce projet sera de représenter graphiquement et approximativement les mouvements de la lune autour de la terre. Nous ne nous intéresserons pas aux mouvements de la terre. Nous considérons que la terre est immobile et que la lune décrit une trajectoire elliptique centrée sur le centre de la terre.

Il s'agira d'abord de modéliser les mouvements de la lune en exploitant quelques éléments de physique et de mathématique classique puis de programmer le rendu de ce satellite et son mouvement autour de la terre.

La bibliothèque utilisée pour effectuer ce projet sera OpenGL avec une programmation en C.

## 1.3 Cahier des charges

- **Contexte**

Notre projet s'inscrit dans un contexte scolaire qui n'a pour but que d'apprendre à faire un vrai projet informatique sans aucun but lucratif et n'ayant également aucun cout prévisionnel. Cependant même les droits et les clauses juridiques ne concernent pas un projet tel que le notre vu que nous sommes dans un cadre uniquement pédagogique.

- **Description du projet**

Notre projet est un logiciel de modélisation du mouvement de la lune autour de la terre. L'objectif de ce logiciel est de représenter graphiquement le mouvement de la trajectoire elliptique de la lune sur le centre de la terre, en utilisant une programmation en C avec une bibliothèque OpenGL, sans oublier d'exploiter des éléments physiques et mathématiques classiques.

On considère que la terre est immobile, elle ne tourne pas autour d'elle-même.

Cette application affichera trois axes x, y et z qui nous serviront de repère et la lune fera une rotation elliptique autour de la terre, le centre du repère. Cette application affichera aussi la masse et la vitesse des deux corps célestes.

L'utilisateur doit utiliser cette application pour mieux comprendre ce phénomène et

pouvoir faire des tests (simulations) avec des masses différentes ou des vitesses différentes des deux corps afin de voir les répercussions sur le mouvement de la lune.

- **Fonctionnalités**

Une fois que notre application est mise en place, l'utilisateur peut faire plusieurs manipulations selon ces choix .Il peut faire des rotations caméras pour mieux visualiser la modélisation, augmenter et baisser la vitesse et la masse de la lune ou la masse de la terre, le logiciel fait aussi les calculs de la trajectoire de la lune en se basant sur des formules physiques (vélocité et position).

Notre application permet aussi de simuler des changements dépendants des masses et leurs vitesses. Vous avez aussi droit à un compteur des jours selon la position de la lune par rapport à la terre tout en respectant la durée de la trajectoire (28 jours approximativement).



## • Les Contraintes

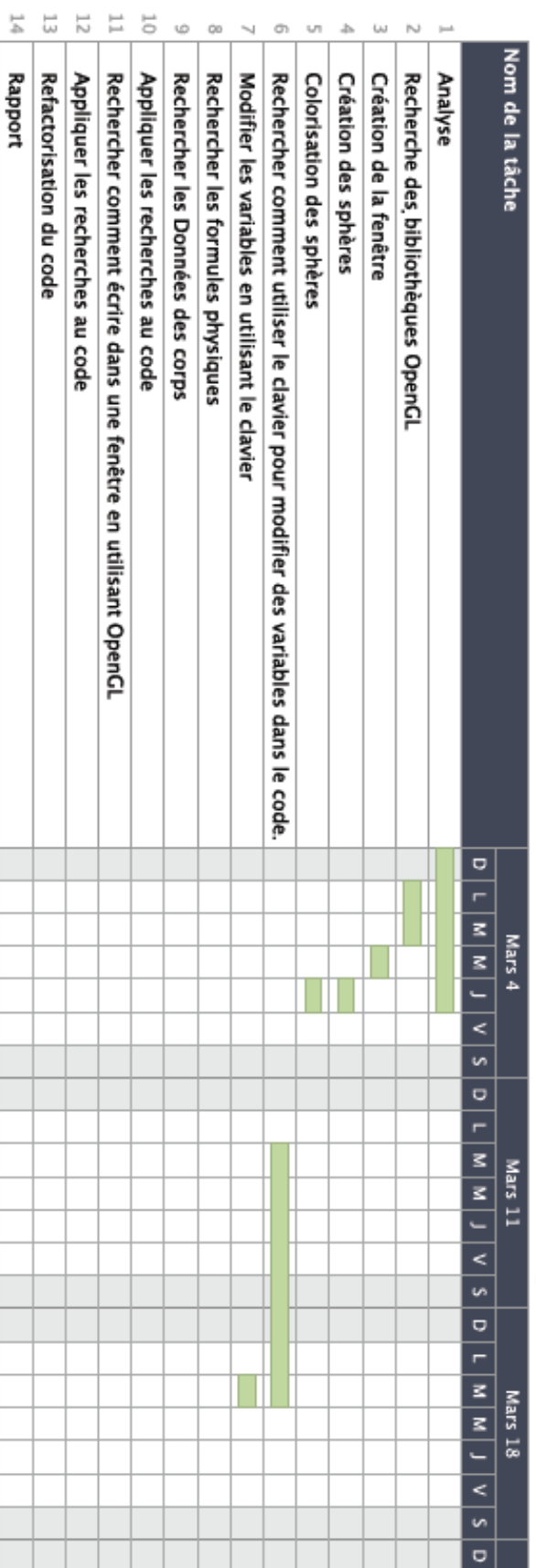
Nous savons tous que tout projet exécuté est fourni à certaines contraintes. Notre projet n'en fait pas l'exception, nous avons rencontré plusieurs contraintes, la plus importante était celle du temps. Nous avons commencé le travail vers mi-mars . Nous ne nous connaissions pas, nos créneaux horaires n'étaient pas compatibles, donc on était amené à se joindre par mail, ce qui n'avancait pas beaucoup dans notre travail.

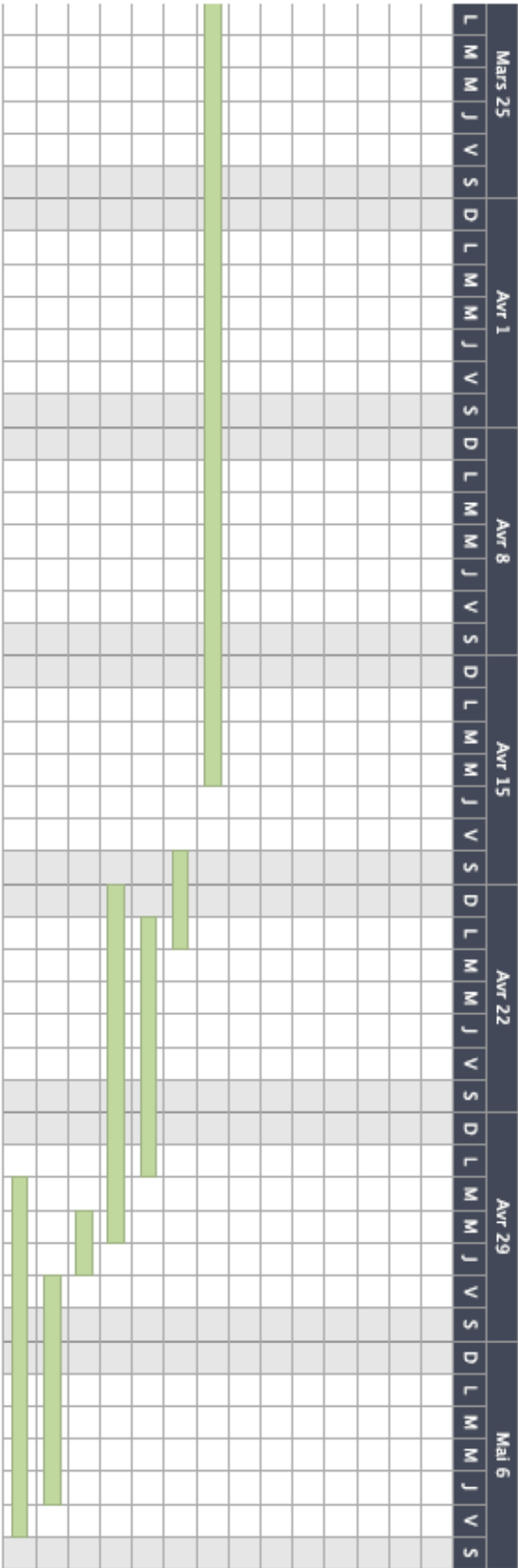
La programmation en C était une contrainte pour moi (Hamza) surtout avec la bibliothèque OpenGL ce qui rendait le travail presque impossible car je n'avais pas acquis cette unité d'enseignement l'année dernière, mais en fin de compte tout s'est bien passé grâce à l'aide d'Isis que je remercie. Pour moi (Isis) c'était plutôt le problème de la langue, pour l'écriture du rapport, pour commenter le code ou pour expliquer mes idées à mon collègue. Le fait d'être que deux pour travailler sur le rapport d'une cinquantaine de pages était un peu compliqué aussi.

Aujourd'hui, nous avons réussi à surpasser toutes ces contraintes grâce à l'aide de notre tuteur, notre patience, et surtout notre détermination.

## Diagramme de GANTT

- ## • Déroulement du projet







## **2. Organisation du projet**

### **2.1 Organisation du travail**

L'organisation du travail était la partie la plus difficile dans ce projet, car pour trouver un créneau horaire qui nous arrangeait tous était presque impossible, du coup nous avons dû employer nos propres moyens comme Facebook, sms et e-mail. Ce problème de créneau nous a été un grand handicap pour la répartition des tâches.

OpenGL est une bibliothèque qui nous a donné du fil à retordre, elle nous a pris beaucoup d'heures de recherche, des problèmes de compatibilité avec le système d'exploitation Windows, qui nous ont amenés en fin de compte à travailler tous les deux sur le Linux. La physique et les notions de géométrie n'étaient pas la partie la moins compliquée non plus vu que ces notions remontent à nos années lycées.

Finalement la partie codage et débogue qui nous a aussi demandé beaucoup de concentration, et surtout de la patience.

### **2.2 Choix des outils de développement**

Les mouvements de la lune autour de la terre est un phénomène physique et mathématique traitant de nombreux calculs scientifiques, d'où le langage de programmation C a été choisi. Ce langage est connu par sa grande capacité de traitement de calculs numériques. Les principales bibliothèques de Stdlib qui ont été utilisées sont :

- « math.h » pour effectuer des opérations mathématiques.
- « stdio.h » pour l'affichage des erreurs et utilisation de quelques fonctions.
- « string.h » pour la manipulation des chaînes de caractères.

Pour le choix de l'éditeur, nous avons choisi en fonction de la facilité et la familiarité que nous avions déjà avec gedit et l'IDE eclipse.

Nous avons choisi le compilateur gcc parce que notre projet a été conçu en C et ce compilateur est open source. Étant donné que les gestionnaires des versions étaient familiers pour l'une des membres du groupe, nous avons adopté le gestionnaire SVN de Google. Le travail est au lien suivant : <http://code.google.com/p/projet-lune/> et est sur GPL v3.

L'utilisation du makefile était une idée ingénieuse, car le makefile est un ensemble de fichiers utilisés par le programme make pour exécuter un ensemble d'actions, comme la compilation d'un projet. Cette méthode est plus facile et plus rapide. Le make compile les fichiers sources. Il nous évite alors de compiler fichier par fichier et évite aussi de recompiler les fichiers non modifiés.

Pour l'élaboration du programme il était nécessaire d'utiliser quelques bibliothèques liées à OpenGL ainsi que cette dernière, je vous les présente :

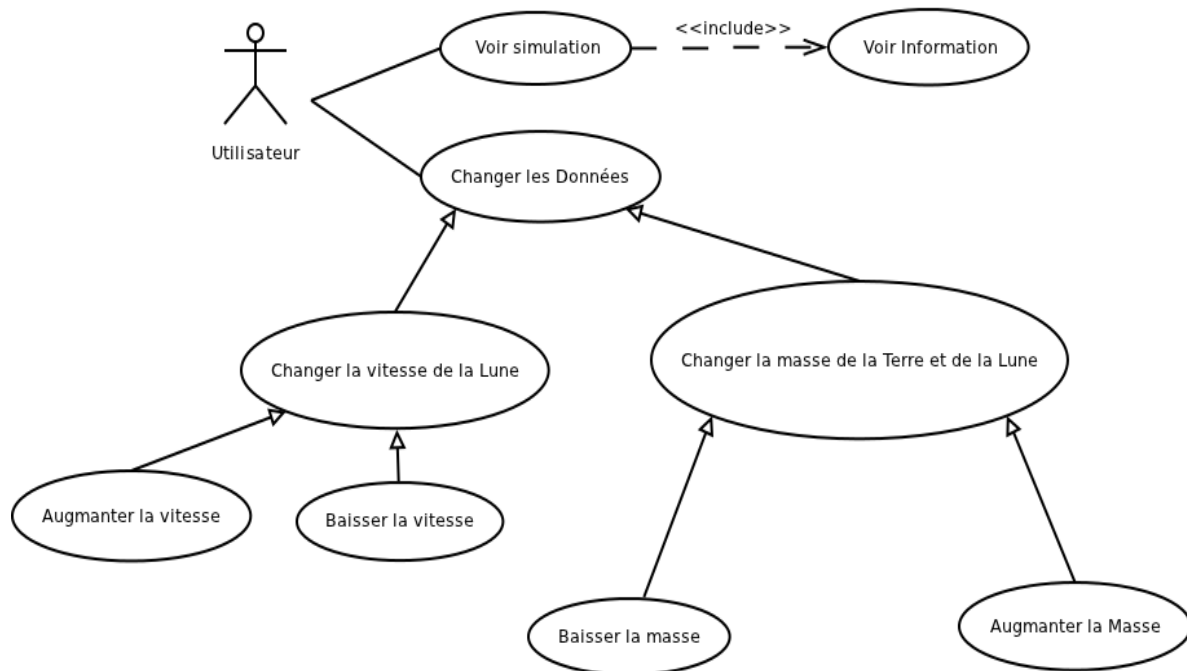
- **OpenGL** (Open Graphics Library) est une spécification qui définit une API multiplateforme pour la conception d'application générant des images 3D et 2D, par conséquent elle est la bibliothèque la plus importante dans notre travail.
- **GLU** (OpenGL Utility Library) est une bibliothèque associée à OpenGL. Elle vient compléter cette dernière en apportant quelques routines pour des opérations de plus haut niveau. GLU a apporté les fonctionnalités suivantes dans notre projet : Gestion de la matrice de la projection pour mettre en place une vue orthogonale ou en perspective, gestion de la matrice de visualisation avec une routine de type « caméra » et gestion d'objets quadriques (sphères).
- 
- **GLUT** (OpenGL Utility Toolkit) est une bibliothèque utilitaire offrant un jeu de routine pour la gestion des fenêtres et les interactions avec le système d'exploitation (gestion clavier, souris, etc.) indépendamment de celui-ci et du gestionnaire de

fenêtres. Elle est essentiellement utilisée pour la création des programmes de démonstrations. Cela explique notre choix d'utiliser cette bibliothèque.

- **GLUI** (OpenGL User Interface Library) est une bibliothèque qui se combine avec celle de GLUT et qui fournit diverses routines pour créer l'interface d'un programme entièrement avec OpenGL. L'interface est alors indépendante du système d'exploitation et du gestionnaire de fenêtres. GLUI nous a servis à écrire sur l'interface du logiciel afin d'afficher les informations nécessaires.

### 3. Analyse du projet

Une représentation basée sur un diagramme UML pour illustrer notre réflexion avant l'étape de réaliser notre programme.



L'analyse est bien claire sur le diagramme présenté. L'utilisateur de notre application a accès à une modélisation du mouvement de la lune autour de la terre avec des informations. Il a aussi le droit de changer des données, changer la vitesse et la masse de la terre et de la lune en augmentant ou baissant ces dernières. Par contre la vitesse de la terre ne peut être modifier car dans notre programme on considère la terre stable, mais on peut modifier sa masse.

Nous avons aussi pensé à ajouter un diagramme de classe mais vu que nous travaillons sur de la programmation impérative dans notre projet, nous avons partagé le travail en trois modules :

- Une partie calcul et physique d'un corps céleste
- Une partie représentation des systèmes célestes
- Une partie OpenGL et la fonction main

Sur l'analyse physique, nous utilisons les formules classiques :

- Formule de gravité universelle

$$\vec{F} = \frac{G \times m_1 \times m_2}{|\vec{r}|^2} \times \hat{r}$$

- Formule d'accélération :

$$\vec{a} = \frac{\vec{F}}{m}$$

- Mouvement :

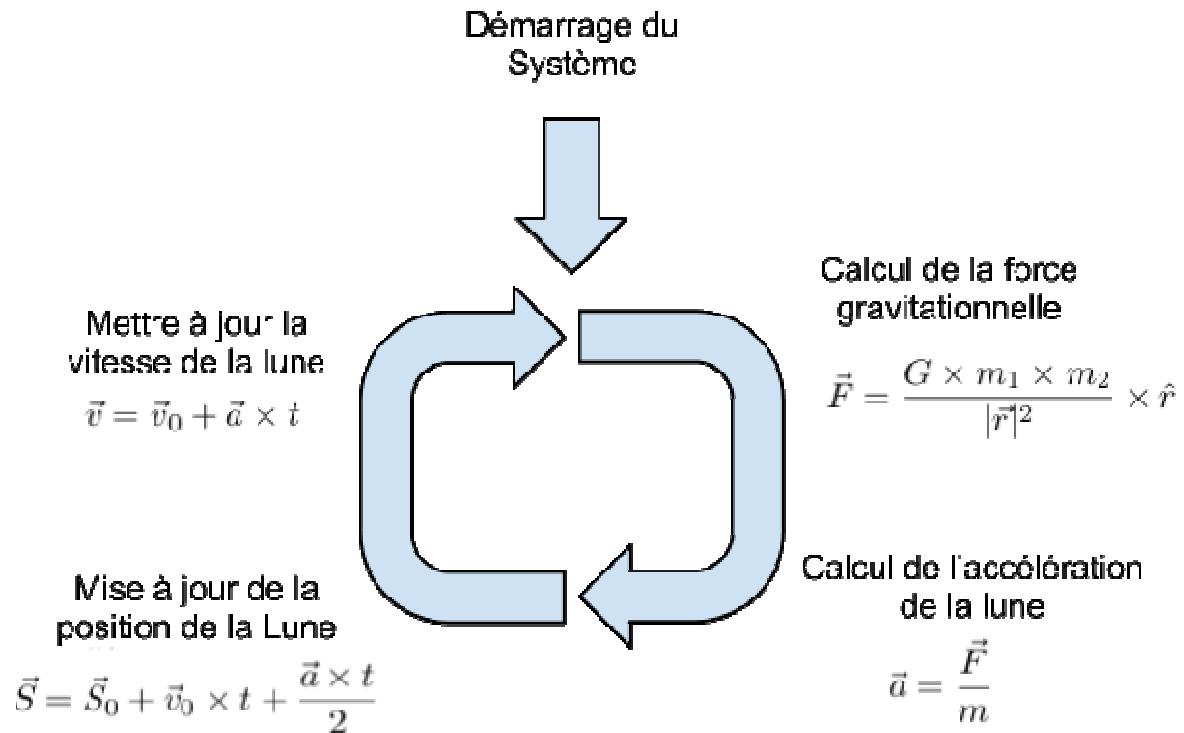
$$\vec{S} = \vec{S}_0 + \vec{v}_0 \times t + \frac{\vec{a} \times t^2}{2}$$

- Vitesse :

$$\vec{v} = \vec{v}_0 + \vec{a} \times t$$



Ainsi, pour le mouvement du système, l'application exécute la boucle suivante:



Pour le démarrage du système il faut entrer les informations initiales: vitesses, masses et positions initiales des corps. Ensuite, pour chaque incrément du temps, il faut calculer la force de la gravité et l'accélération de la lune puis on doit mettre à jour leurs positions et leurs vitesses.



## 4. Développement

La première partie du développement était plutôt de la recherche et la découverte d'OpenGL, afin de pouvoir bien exploiter les fonctions de cette librairie dans notre développement. Dans l'étape suivante, nous avons commencé par créer une fenêtre qui représentera notre application en choisissant une couleur pour l'arrière plan (background), puis nous créons des sphères tout en leurs donnant deux couleurs différentes. Nous créons aussi une fonction pour que l'une des sphères (la lune) fasse un mouvement autour de l'autre.

Pour une modélisation meilleure nous implémentons la modification des variables (vitesses, masses) grâce au clavier. Comme nous le savons tous, le mouvement de la lune autour de la terre repose sur des formules physiques, nous appliquons ces formules et d'autres données de la lune et la terre dans un fichier code nommé « fisca.h ». Maintenant qu'on a deux sphères avec deux couleurs différentes, les lois physiques et la rotation sont respectées, il ne nous reste plus qu'à nommer les deux sphères terre et lune.

Pour cela, nous étions amenés à faire des recherches pour apprendre à afficher une écriture sur une application en utilisant OpenGL. Grâce à aux résultats de nos recherches, on implémente l'affichage des commentaires sur notre application.

Pour représenter les corps célestes, on a créé la structure suivante, qui contient les informations générales concernant notre simulation.

```
typedef struct {  
    double rayon;  
    char nom[21];  
    double masse;  
    double position[3]; // vectoriel  
    double vitesse[3]; // vectoriel  
    double couleur[3]; // couleur en RGB  
} Planet;
```

Comme les corps sont situés dans l'espace et la vitesse est une grandeur vectorielle, nous utilisons les tableaux en C pour les représenter. Nous les avons utilisés aussi pour les composants RGB de la couleur du corps.

## 1. Créer une Fenêtre

```
glutInit(&argc, argv);  
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);  
glutInitWindowSize(800, 600);  
glutCreateWindow("Modélisation, rendu et mouvement de la lune autour de la terre :");
```

Pour créer une fenêtre il suffit d'initialiser la bibliothèque « glut » et utiliser les fonctions fournies.

## 2. Mettre une couleur en arrière plan

```
glClearColor(0, 0, 0, 0.0);
```

Nous pouvons choisir une couleur en RGB et un *alpha* (transparence) pour l'arrière plan en utilisant cette fonction.

## 3. Créer les sphères

```
glutSolidSphere(unPlanet.rayon / rayonCenter, 25, 25);
```

Les sphères sont créées par cette fonction de la bibliothèque « glut » qui reçoit comme paramètres le rayon de la sphère et le nombre des cercles verticaux et horizontaux. Pour une sphère plus grande, il suffit d'ajouter des cercles.

On a divisé le rayon du corps céleste à dessiner par le rayon du corps au centre du système. Ainsi le rayon de la terre divisé par lui-même pour avoir 1, et la lune, qui est trois fois plus petite, aura approximativement un rayon de  $\frac{1}{3}$ .

#### 4. Colorier les sphères

```
glColor3f(unPlanet.couleur[0], unPlanet.couleur[1], unPlanet.couleur[2]);
```

Pour colorier les sphères il faut mettre la couleur souhaitée avant de les dessiner. Cette couleur est en codification RGB (le « 3f » signifie qu'on utilise les valeurs entre 0 et 1 pour les composants, pas 0 et 255).

#### 5. Créer le mouvement de la lune

Pour le mouvement du système, il faut implémenter la boucle vue dans la partie analyse. Le premier pas est d'initialiser la terre et la lune avec les données physiques collectées dans les étapes de recherche.

```
void init_terre(Planet* terre)
{
    strcpy(terre->nom, "Terre");
    terre->couleur[0] = 0;
    terre->couleur[1] = 0;
    terre->couleur[2] = 1.0;

    terre->masse = 5.988e24;

    terre->vitesse[0] = 0;
    terre->vitesse[1] = 0;
    terre->vitesse[2] = 0;

    terre->position[0] = 0;
    terre->position[1] = 0;
    terre->position[2] = 0;

    terre->rayon = 6.378e6;
}
```

```
void init_lune(Planet* lua)
{
    strcpy(lua->nom, "Lune");
    lua->couleur[0] = 1.0;
    lua->couleur[1] = 1.0;
    lua->couleur[2] = 1.0;

    lua->masse = 7.35e22;

    lua->vitesse[0] = 0;
    lua->vitesse[1] = 0;
    lua->vitesse[2] = -1023.0;

    lua->position[0] = 3.844e8;
    lua->position[1] = 0;
    lua->position[2] = 0;

    lua->rayon = 1.738e6;
}
```

On utilise la fonction « glutTimerFunc() » pour faire la boucle principale. Avec cette fonction on définit un temps en Mili secondes et on appelle une autre fonction après ce temps. Notre fonction appelée est la « Timer() ».

```
void Timer(int value) {  
  
    passerLeTemps();  
    desenha();  
  
    // j'appelle toujours cette fonction pour faire une boucle .  
    glutTimerFunc(10, Timer, 0);  
}
```

Dans cette fonction on appelle la fonction « passerLeTemps() », définie et implémentée dans les fichiers « système », qui permet d'incrémenter le temps dans le système. Après on appelle la fonction « desenha() » pour dessiner le système et en fin nous mettons la fonction « glutTimerFunc() » pour appeler la fonction « Timer() » à nouveau.

```
void passerLeTemps()  
{  
    //je calcule la force  
    double force[3];  
    forceG(lune, centre, force);  
  
    // et le mouvement  
    temp += calcMouvement(&lune, force);  
}
```

Dans la fonction « passerLeTemps() » il y a toute la logique de l'écoulement du temps du système. Premièrement on calcule la force de gravité avec la fonction « forceG() » et après on l'utilise pour calculer le mouvement dans la fonction « calcMouviment() », que retourne le temps écoulé.

```
void forceG (Planet p1, Planet p2 , double force[])
{
    const double G = 6.67e-11; // Constant de la gravitation
    double r[3]; // vector du planète1 au planète2
    double d;
    r[0]= p2.position[0] - p1.position[0];
    r[1]= p2.position[1] - p1.position[1];
    r[2]= p2.position[2] - p1.position[2];

    // distance ou norme de r
    d = sqrt(r[0]*r[0]+r[1]*r[1]+r[2]*r[2]);

    r[0] = r[0]/d;
    r[1] = r[1]/d;
    r[2] = r[2]/d;

    // calcul de la force pour chaque composant vectorielle
    force[0]= (G*p1.masse*p2.masse)/(d*d)* r[0];
    force[1]= (G*p1.masse*p2.masse)/(d*d)* r[1];
    force[2]= (G*p1.masse*p2.masse)/(d*d)* r[2];
}
```

La « forceG() » est le codage de la fonction de gravité classique décrit dans la partie analyse. Elle calcule la force résultante exercée sur les corps « p1 » et « p2 » et met le résultat dans le tableau reçu « force[] ».

```

int calcMouviment(Planet* p2, double force[])
{
    double a[3];
    const double t = 900;

    //calcul de l'acceleration
    a[0]= force[0]/ p2->masse;
    a[1]= force[1]/ p2->masse;
    a[2]= force[2]/ p2->masse;

    // Calcul de la nouvelle position
    p2->position[0] = p2->position[0]+ p2->vitesse[0] * t+ a[0]*(t*t)*1/2;
    p2->position[1] = p2->position[1]+ p2->vitesse[1] * t+ a[1]*(t*t)*1/2;
    p2->position[2] = p2->position[2]+ p2->vitesse[2] * t+ a[2]*(t*t)*1/2;

    // Calcul de la nouvelle vitesse
    p2->vitesse[0] = p2->vitesse[0]+ a[0]* t;
    p2->vitesse[1] = p2->vitesse[1]+ a[1]* t;
    p2->vitesse[2] = p2->vitesse[2]+ a[2]* t;

    // ont retourne le temps utilisé pour les calculs
    return t;
}

```

Pour calculer le mouvement on utilise la force, la masse, la vitesse et la position précédente. On met à jour la position et la vitesse du corps et on retourne le temps utilisé dans les calculs.



## 6. Implémenter l'utilisation du clavier pour modifier des variables dans le code.

La bibliothèque « glut » nous permet de définir des fonctions pour le traitement du clavier.

```
glutSpecialFunc(teclas_especiais);  
glutKeyboardFunc(teclas_normais);
```

Dans notre cas, on définit les fonctions « teclas\_especiais() » et « teclas\_normais() » pour le traitement des clés spéciales et normales respectivement.

```
void teclas_normais(unsigned char key, int x, int y) {  
    switch (key) {  
        // 27 est la clé 'echap' à la table ascii (elle ferme le logiciel)  
        case 27:  
            exit(0);  
  
        case 'r': //je redémarre le logiciel avec 'r'  
            init_systeme();  
            break;  
    }  
}
```

Dans « teclas\_normais() » on ferme le logiciel avec la clé « echap » et on le redémarre avec la clé « r ». Pour les modifications des variables du système on fait le traitement des clés spéciales dans la fonction suivante.

## 7. Ecrire dans la fenêtre en utilisant OpenGL

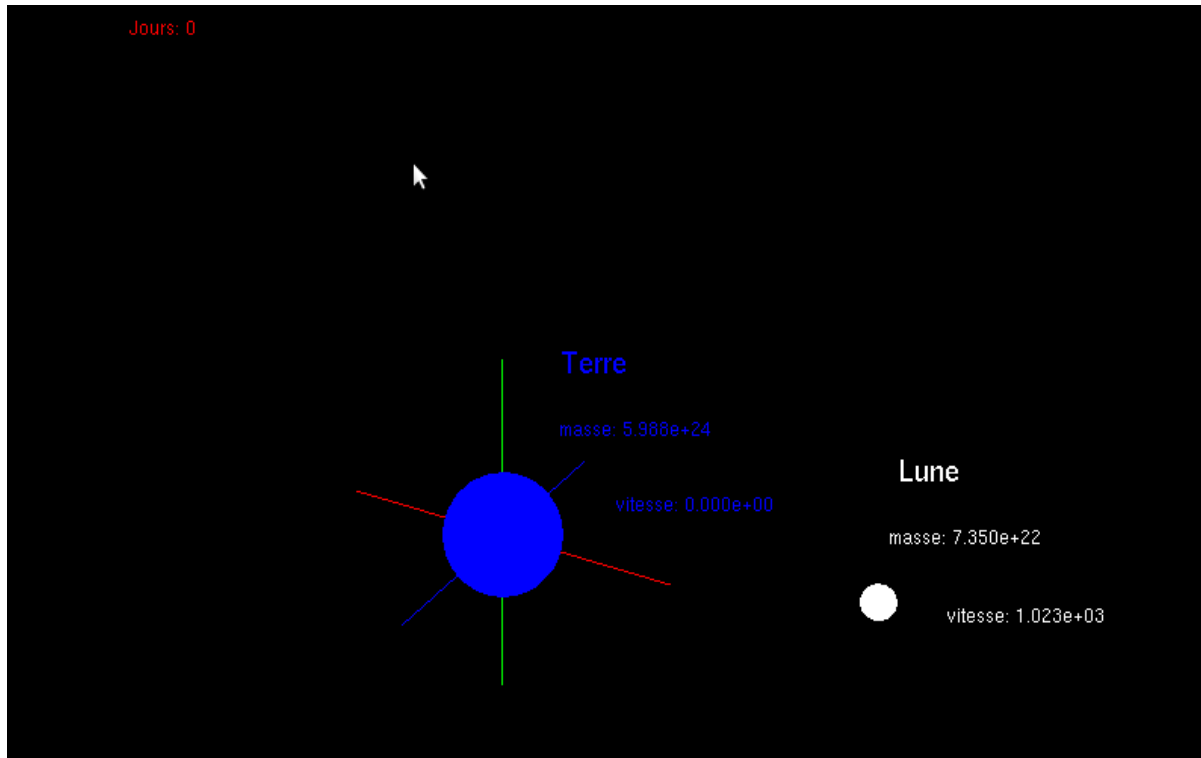
Grâce à nos recherches sur internet, on a vu que la bibliothèque « GLUT » peut être utilisée pour écrire sur la fenêtre. Exemple:

```
sprintf(nome, "%s ", unPlanet.nom);  
  
_glutBitmapString(GLUT_BITMAP_HELVETICA_18, nome);  
glRasterPos2i(unPlanet.rayon / rayonCenter, unPlanet.rayon / rayonCenter + 1.40);
```

On a utilisé « sprintf() » pour formater une chaîne de caractères et placer le résultat dans la variable « nome ». la fonction « \_glutBitmapString() » reçoit comme variable le style de la lettre et la string. Et « glRasterPos2i() » reçoit la position désirée.

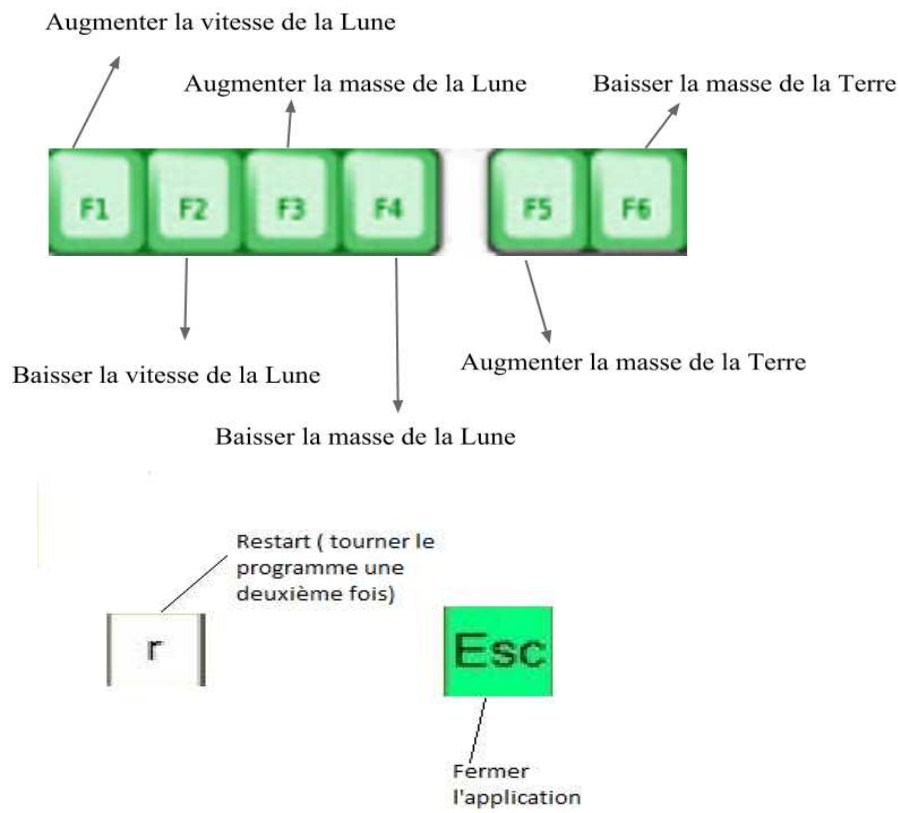
## 5. Manuel d'utilisation

A l'exécution du programme, l'écran d'accueil de notre application ressemble à l'image ci dessous:



Comme vous pouvez le remarquer l'application affiche la planète terre, sa masse, sa vitesse en bleu (on considère que la terre est stable donc sa vitesse est toujours nulle) et la lune, sa masse, sa vitesse en blanc. En haut de la fenêtre en rouge, on remarque le compteur des jours que la lune met pour faire un tour sur la terre.

Pour une simulation et modélisation meilleure, l'utilisateur peut faire des modifications des données grâce aux touches du clavier:

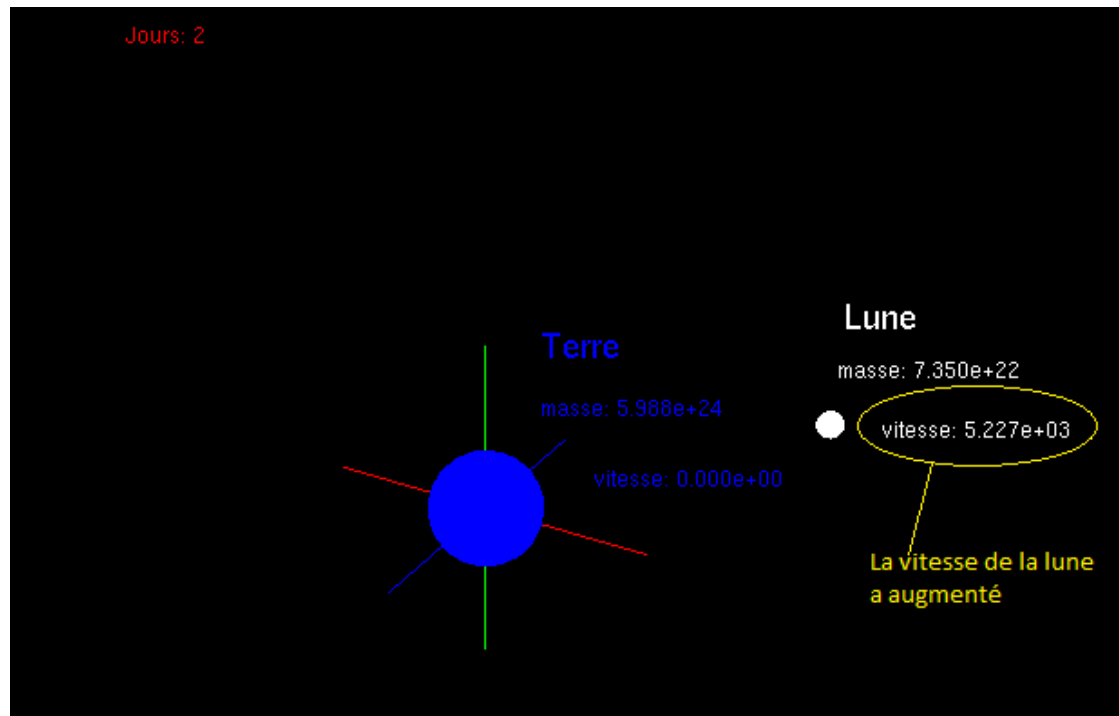


Pour une meilleure visualisation:



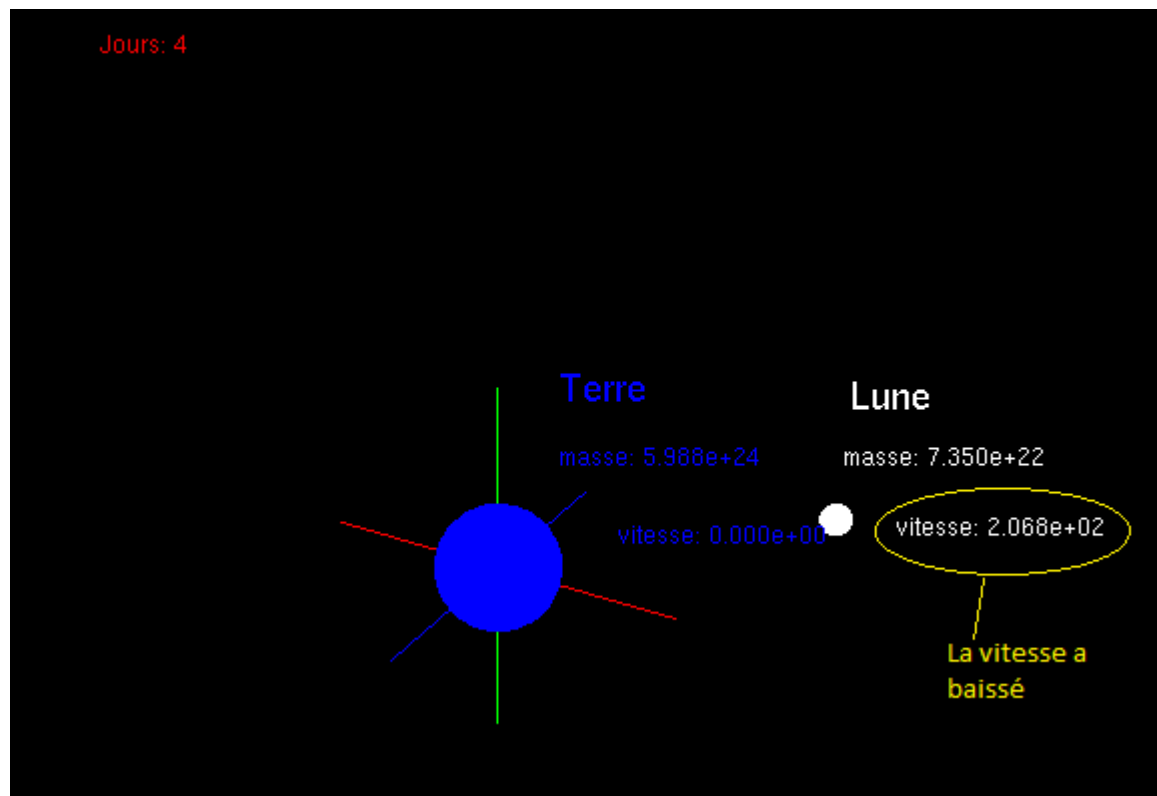
Les flèches du clavier servent à changer les angles de la caméra.

On regarde qu'est ce qui se passe quand on appuie sur la commande F1 :



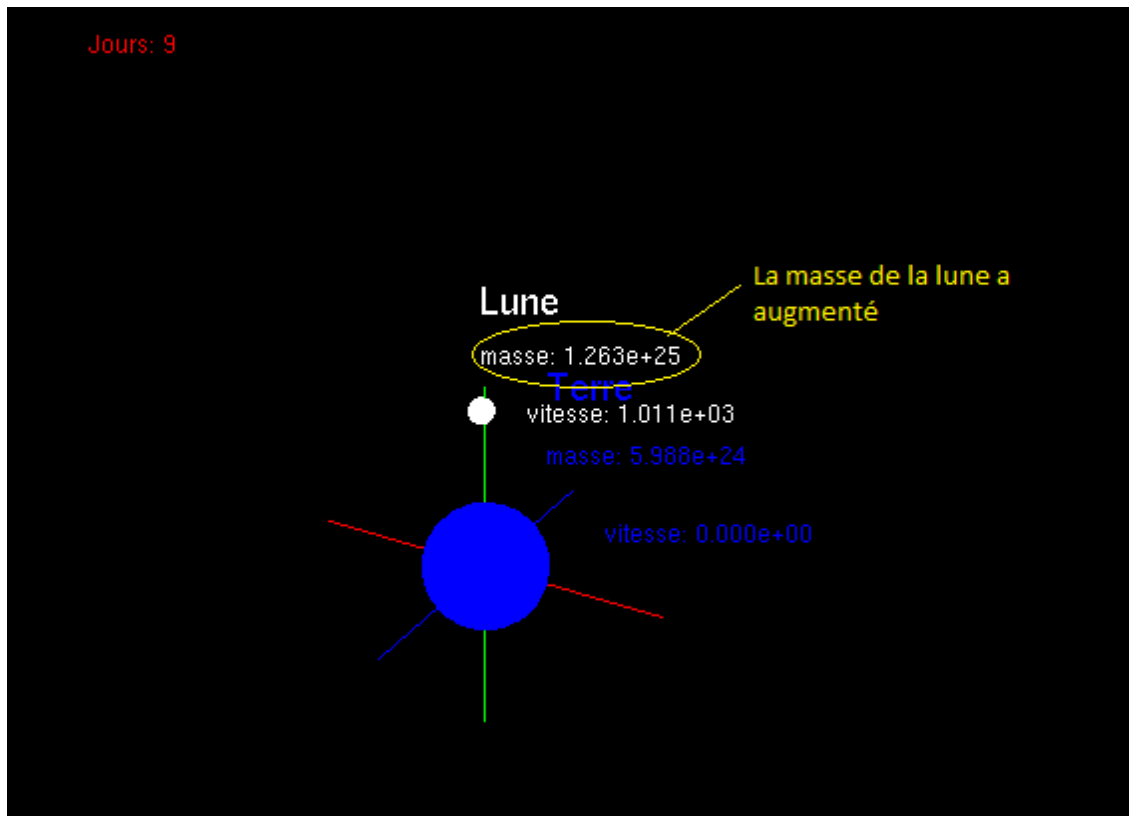
Si la vitesse augmente, nous verrons sur la simulation que la lune continue tout droit en sortant de sa trajectoire jusqu'à disparaître de la fenêtre.

## La commande F2

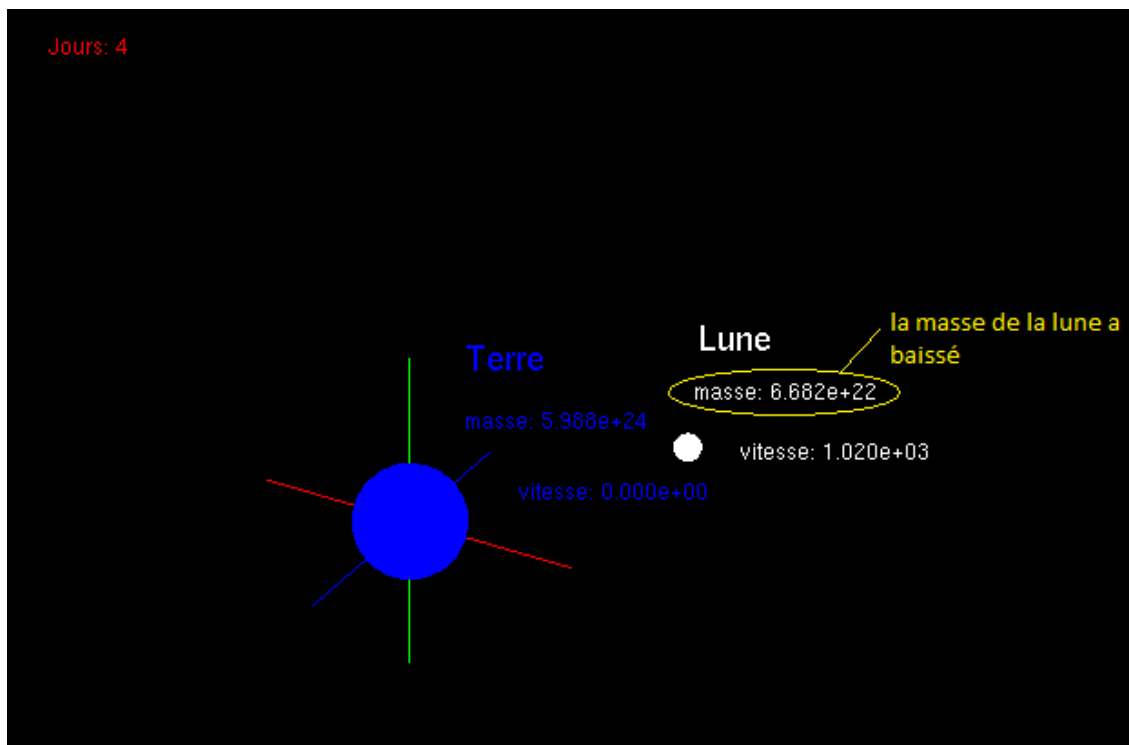


Si la vitesse baisse, nous verrons que la lune se dirige directement vers la terre et disparaît. Ce qui explique une collision entre les deux corps célestes.

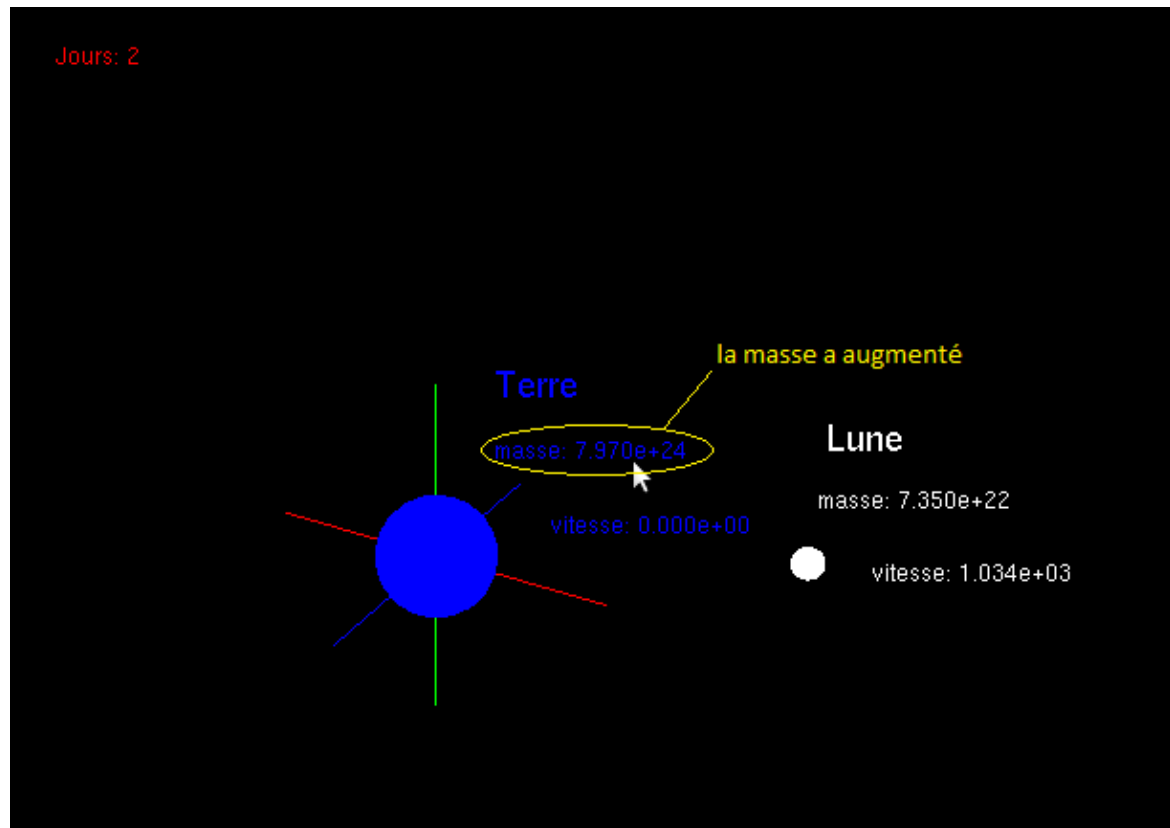
### La commande F3



### La commande F4

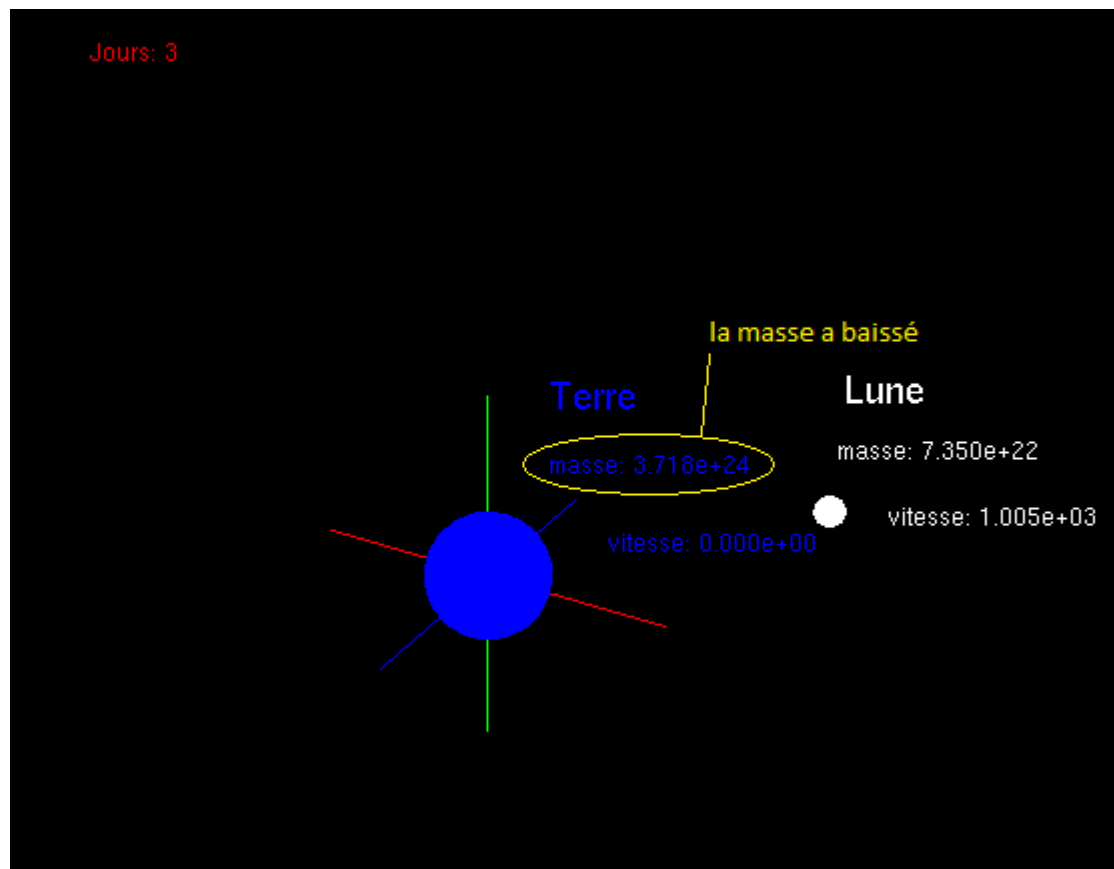


## La commande F5





## La commande F6





## **6. Perspectives et conclusions**

### **6.1 Perspectives**

Pour les futures versions de ce projet, nous espérons voir des applications plus avancées en matière de modélisation, une interface graphique plus pratique et plus réaliste. Il sera intéressant d'ajouter tout le système solaire par exemple, en appliquant des textures ainsi que de la physique pour chaque corps. En ce qui concerne l'interface graphique : ajouter des boutons de commandes, des slides pour tout ce qui peut croître et décroître, un fond d'écran de l'espace avec des étoiles.

La meilleure amélioration serait de pouvoir modéliser le mouvement de la terre autour d'elle même sans déstabiliser la modélisation de tout le système, chose très compliquée à faire, car le fait que toutes les données soient approximatives rend le système facile à se déstabiliser.

### **6.2 Conclusions**

L'application réalisée est fonctionnelle et prête à l'emploi. Le mouvement elliptique de la lune autour de la terre est bien présent dans la modélisation, les lois physiques et mathématiques sont belles et bien respectées, les valeurs réels approximatives des données aussi, la simulation des changements des masses, des vitesses planétaires sont modifiables par l'utilisateur.

Un compteur des jours selon la position de la lune par rapport à la terre tout en respectant l'ellipse (28 jours approximativement) est mis en place aussi. La manipulation de l'application est facilitée grâce au manuel d'utilisation. L'utilisateur peut changer l'angle de la caméra ou même augmenter et baisser la vitesse, la masse des planètes grâce au clavier.

Les outils choisis nous conviennent bien, parce que la majorité de ces derniers est déjà vue dans le programme de la première et deuxième année de licence informatique. La bibliothèque OpenGL était la grande découverte dans notre projet donc il nous a fallu plus de temps pour s'y familiariser et s'y adapter. On peut aussi ajouter qu'elle est très puissante en matière de graphisme. L'analyse, nous pensons qu'elle était suffisante vue que notre application fonctionne convenablement et obéit aux demandes du sujet.

Ce projet a été très bénéfique pour nous, c'était une expérience très enrichissante car nous avons pu travailler sur les différentes phases de la réalisation d'une application réelle de modélisation, en partant des besoins à la concrétisation des solutions répondant à ceux-ci.

Pour commencer, nous avons pu acquérir de nouvelles compétences en travaillant en collaboration entre deux étudiants qui ne se connaissent pas, de pays différents, de cultures différentes et même un niveau de compétence différent, ce qui nous a permis d'apprendre beaucoup de chose l'un de l'autre, de s'entraider, de surpasser les difficultés rencontrées tout au long de ce parcours et enfin réaliser un travail considérable en un temps remarquable.

--