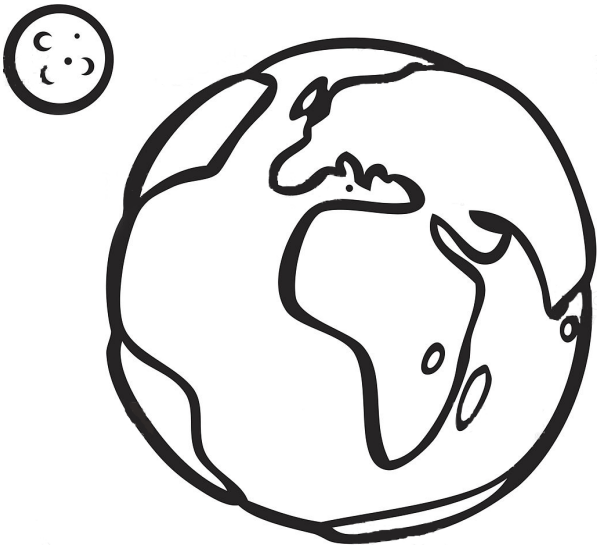
Rapport du Projet Terre Lune

HLIN405 - TERL2



Année universitaire 2016/2017

La Terre est séparée d'une distance de 384 800 km de son satellite naturel, la Lune.

Cette dernière possède un rayon de 1 737 km. Il existe entre les deux planètes une force d'attraction gravitationelle qui est à l'origine du mouvement elliptique de la Lune.
Sa période de révolution (rotation autour de la Terre) est de 27,3 jours. De plus, cet astre tourne sur lui-même en 27,3 jours également.

La Terre est l'une des plus grandes planètes telluriques du système solaire. Avec un rayon de 6 371 km, elle est environ 3,5 fois plus imposante que la Lune.
Sa période de révolution autour du Soleil est de 365 jours.

Notre projet consistait à réaliser un programme qui modélise le système "Terre/Lune".

Projet réalisé par:
— BARRY Mamadou Djouldé
— DEDEBANT Louis
— DIABATE Amara
— QUENETTE Christophe
— SAI Ismael

Sous la tutelle de Mountaz Hascoët dans le cadre du TER L2 informatique

Table des matières

Chapitre 1

Objectif

1.1 Cahier des charges

Le but de ce projet était la modélisation du mouvement de la Lune gravitant autour de la Terre.

Pour cela, nous avons eu recours aux fonctions permettant les calculs d'images 3D de la bibliothèque Open Graphics Library.

Concernant le langage de programmation, nous avons favorisé l'utilisation d'objets plutôt que de stocker les données dans des structures. L'application a donc été développée en C++. Ce fut l'occasion de pratiquer ce langage, ce qui nous a également motivé à faire ce choix.

Tout au long de l'avancement du programme, nous avons exporté les différents fichiers sur les serveurs de la faculté à l'aide de GitLab. En plus de nous garantir un accès au code, cela nous a permis d'essayer de l'améliorer ou de le corriger chacun de notre coté, à partir de branches individuellement affectées.

Le cahier des charges nous imposait plusieurs contraintes :

— Écrire un programme qui calcule le mouvement de la lune autour de la terre.
— Afficher la Lune et la Terre avec la rotation de la Lune autour de la Terre.
— Éclairage : ajouter une source lumineuse à l'infini
— Faire en sorte que les paramètres du modèle puissent être facilement modifiés
— Choisir la fonctionnalité supplémentaire à ajouter : ajout de textures, simulations sur des modifications des paramètres : vitesse, masses, etc.

Le projet est consultable sur gitlab à l'adresse suivante :
`https://gitlab.info-ufr.univ-montp2.fr/e20140034671/TerreEtLune.git`

1.2 Prémices

Dès que nous avons eu confirmation du sujet de TER, nous nous sommes immédiatement mis à faire des recherches sur la Terre, la Lune et les mouvements qui leur sont associés.

Suite au premier rendez-vous avec notre encadrante Mme Hascoët, nous avons obtenu des renseignements sur le déroulement du projet. Elle nous a fourni les rapports des années précedentes.

Après une brève étude de ces rapports, des consgines, et grace à nos recherches personnelles, nous avons décidé de modeliser ce problème sous forme de classes.

Nous avions déjà quelques idées de classes en tête, telle qu'une classe "Planète" qui permettrait de modéliser à la fois la Terre et la Lune sachant qu'elles partagent beaucoup de caractéristiques. Plusieurs attributs d'une planète sont representés sous forme de vecteur, ce qui nous a conduit à imaginer une classe représentant un tel objet.

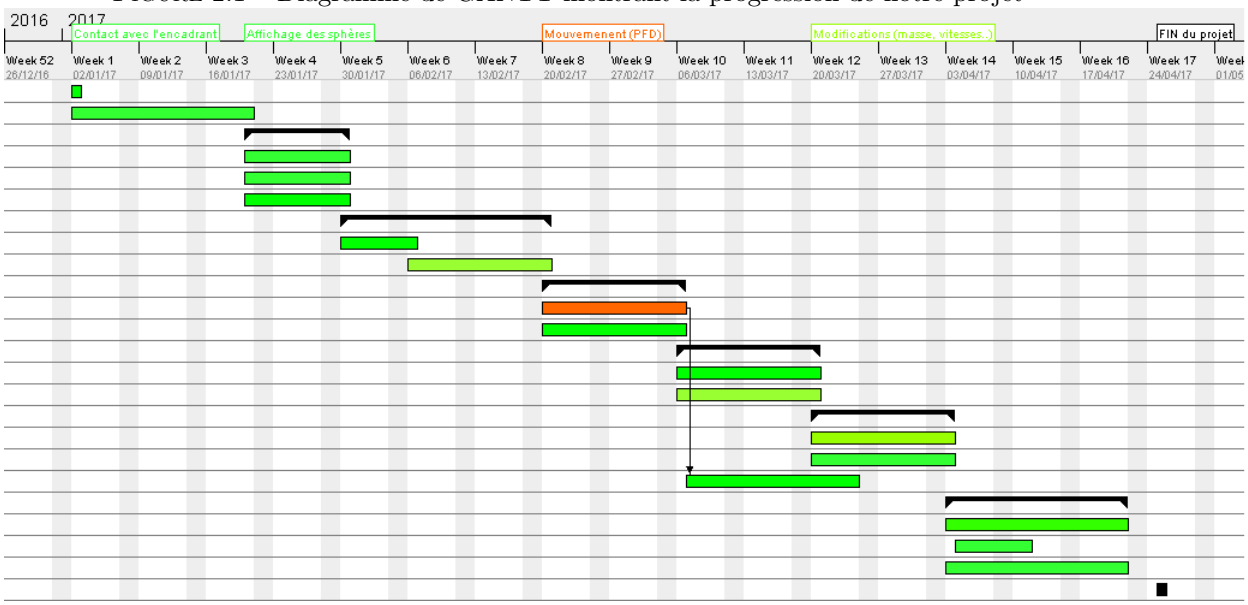
Avec tous ces éléments, nous avons réalisé un croquis de diagramme UML qui nous a permis de poser à plat nos idées et des les ordonner.

Chapitre 2

Conduite de projet

2.1 Diagramme de GANTT

FIGURE 2.1 – Diagramme de GANTT montrant la progression de notre projet



Avec le professeur encadrant, il a été convenu d'avoir un rendez-vous tous les 15 jours.
On a donc effectué les tâches suivantes sous quinzaines, représentées graphiquement par les intervalles noirs.

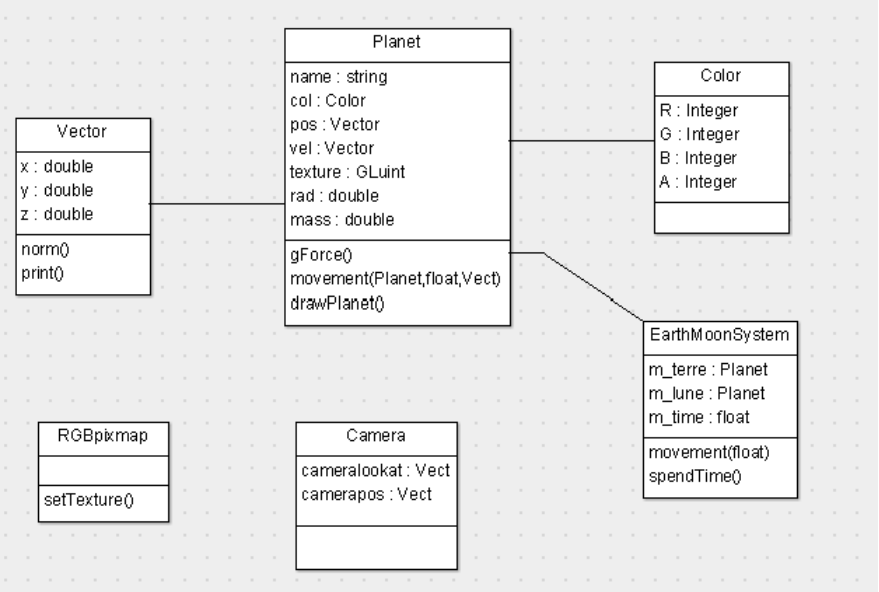
- 20/01 - 30/01 :
  - Affichage des sphères
  - Recherche sur les orbites circulaires uniformes
  - Analyse des rapports (versions antérieures)

- 30/01 - 20/02 :
  - Calcul sur excel des différentes positions de la lune
  - Implémentation d'un mouvement circulaire uniforme
- 20/02 - 06/03 :
  - Mouvement (PFD)
  - Fonction appliquant la force gravitationnelle

— 06/03 - 20/03 :
  — Documentation du code (Doxygen)
  — Gestion de la caméra
— 20/03 - 04/04 :
  — Modification (Masse et vitesse)
  — Textures
  — Suite de l'implémentation du mouvement (PFD)
— 04/04 - 21/04 :
  — Éclairage dans l'univers
  — Tests unitaires
  — Rédaction du rapport

2.2 Diagramme UML

FIGURE 2.2 – Diagramme de classes UML récapitulatif



Comme dit précedemment, des classes ont été conçues pour faciliter le développement de la classe "Planet" :

— classe vector :

Permet de représenter les données à trois valeurs (x,y,z) sous forme d'un objet vecteur.

— classe color :

Permet de colorier les sphère pour le mode schématique .

— classe RGBpixmap :

Permet l'ajout de textures sur les sphères.

Et d'autres classes ont été créées pour le développement du système :

— classe Caméra :

Permet de visualiser la simulation sous différents angles .

— classe EarthMoonSystem :
Cette classe veille au bon déroulement de l'execution (initialise deux objets "Planet" avec des données réalistes et s'occupe de la gestion temporelle).

Concernant la classe "Planète" elle possède les méthodes responsables du calcul de mouvement lunaire.

Pour plus de précisions, le code a été documenté avec Doxygen (disponible sur le gitlab).

Chapitre 3

Développement

3.1 Compilation

Afin de compiler le programme, nous avons fait le choix d'utiliser un makefile en vue d'automatiser ce processus.

```
NAME  =  tel
CXX   =  g++
LDFLAGS  =  -lglut -lGL -lGLU -lm
SRCS  =  ../src/main.cpp              \
         ../src/planet.cpp            \
         ../src/vect.cpp              \
         ../src/color.cpp             \
         ../src/earthMoonSystem.cpp   \
         ../src/RGBpixmap.cpp         \
         ../src/camera.cpp
CXXFLAGS=   -W -Wall -ansi -pedantic -I./inc -std=c++11
OBJS  =  $(SRCS:.cpp=.o)
all: $(NAME)
$(NAME): $(OBJS)
   $(CXX) -o $(NAME) $(OBJS) $(LDFLAGS)
clean:
   rm -f $(OBJS)
fclean: clean
   rm -f $(NAME)

re: fclean all
```

Ce Makefile produit un exécutable nommé tel, acronyme de Terre Et Lune. Il prend logiquement pour cible les fichiers du répertoire src (contenant les fichiers .cpp) et utilise les flags nécessaires.

3.2 Représentation du système

Le système d'execution sera représenté dans le progamme principal à l'aide d'un objet de la classe "earthMoonSystem" ; qui prend en attribut deux planètes, une camera, et les entiers permettant la gestion du temps.

```
class EarthMoonSystem{ private:
    Planet m_terre, m_lune;
    unsigned int m_time, _deltaT;
    Camera m_cam;
};
```

"spendTime()" appelle la méthode calculant la position de la Lune et gère le temps écoulé à l'aide d'un incrément constant m_deltaT.

```
1 void EarthMoonSystem::spendTime(){
2     m_lune.movement(m_terre, m_deltaT, m_lune.gForce(m_terre));
3     m_time += m_deltaT;
4 }
```

3.3 Calul du mouvement

Pour que la trajectoire de la lune soit la plus fidèle possible à la réalité il fallait que celle-ci soit une ellipse. Cependant, pour que la trajectoire soit précise des calculs physiques relativement complexes étaient nécessaires. En voici une trace :

$$\sum \vec{F_{ext}} = m\vec{a} \tag{3.1}$$

$$u(\theta) = A\cos(\theta + \phi) + \frac{\alpha}{r_0^4 \dot{\theta}_0} \tag{3.2}$$

A partir du PFD (3.1), de l'équation du ressort (3.2) et de la conservation du moment cinétique on obtient l'équation de la trajectoire suivante :

$$r(\theta) = \frac{1}{A\cos(\theta + \phi) + \frac{\alpha}{r_0^4 \dot{\theta}_0^2}}$$

Avec $r(\theta)$ la position en fonction de l'angle θ.
Ce qui nous représenterait le mouvement elliptique désiré.

3.4 Options supplémentaires

Nous avons pris l'initiative d'ajouter au programme plusieurs fonctionnalités :

- Une caméra mobile,
- Une touche permettant de ré-initialiser la simulation.
- Un mode permettant d'avoir une vision schématique de la rotation (selon l'axe Z) c'est a dire où le tracé de la trajectoire est plus parlant et avec l'ajout des caracteristiques des planètes et du temps
- La rotation de la Terre sur elle même en 24h (codée de façon réaliste c'est à dire qu'il y a cohérence temporelle : pour une révolution entière de lune, la Terre aura tournée sur elle-même 27 fois)
- L'ajout de textures sur la Terre, la Lune et sur la skybox représentant l'univers.

FIGURE 3.1 – Mode schématique

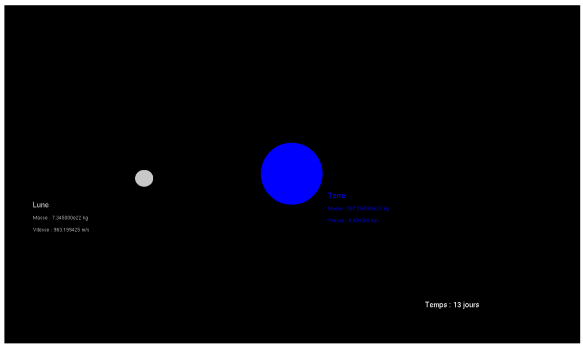
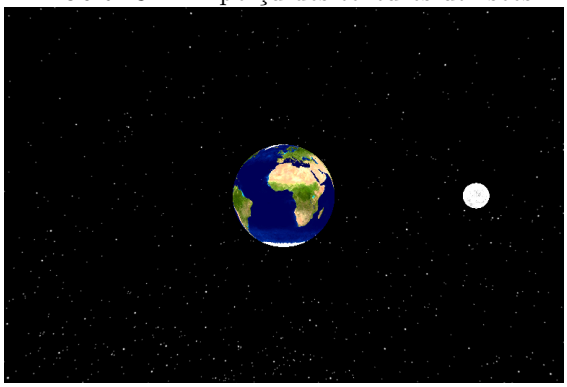


FIGURE 3.2 – Aperçu des textures utilisées



Chapitre 4

Conception

Concernant l'organisation du système de fichiers, nous avons répertorié les fichiers de telle sorte :

— les fichiers sources dans «src»,

— les fichiers d'en-tête dans «inc»,

— les images pour les textures dans «bitmaps»,

— les fichiers pour la documentation Doxygen dans «doc»

— le Makefile dans «build».

— des fichiers utiles dans «annexe».

On décrira ci-après les choix que nous avons pris lorsque nous étions confrontés à des problèmes, suivis des solutions que nous avons adopté pour y remédier.

4.1 Problèmes rencontrés

Lors de la conception du code nous avons rencontrés certains problèmes :

4.1.1 Camera

Les appels des accesseurs de la caméra ne fonctionnaient pas et par conséquent la position de la caméra ne changeait pas.

```
1    Vect Camera::getCP() {return camerapos;}
2    Vect Camera::getLP() {return cameralookat;}
3    void Camera::setCP(Vect pos) {camerapos = pos;}
4    void Camera::setLP(Vect la) {cameralookat = la;}
```

4.1.2 Modification de la vitesse de la lune

La methode permettait de modifier la vitesse, malheureusement la modification n'était pas sauvegardée.

```
void mulVel(double d){

        getVel().setX( getX()* d);
        getVel().setY( getY()* d);
        getVEl().setZ( getZ()* d);
}
```

4.1.3 Calcul du mouvement

La representation du mouvement de la Lune à partir du calcul entier issu du PFD décris dans la phase de developpement etait trop compliqué à implémenter. Comme nous etions bloqués, nous avons pris rendez-vous avec Monsieur Dorignac.

4.2 Solutions trouvées

4.2.1 Camera

Pour regler ce problème nous avons fait une refonte totale de la classe camera. Au lieu de passer par des vecteurs nous sommes passés par plusieurs flottants représentant chacun une coordonée (x, y et z).

```
  float c_x;
  float c_y;
  float c_z;

  float l_x;
  float l_y = 1.0f;
  float l_z;

  float u_x = 0.0f;
  float u_y = 1.0f;
  float u_z = 0.0f;

  float angle;
```

De plus, l'accesseur en lecture de la camera du système retourne la référence de l'objet.

```
Camera *EarthMoonSystem::getCam()
{
    return &m_cam;
}
```

4.2.2 Modification de la vitesse de la Lune

Les objets manipulés par les accesseurs n'etaient pas les mêmes. C'était une copie des objets initiaux. Nous sommes donc passés par les références de l'attribut vitesse de la lune, afin de modifier les objets eux mêmes.

```
Vect *Planet::getVel(){ return &vel; }
```

4.2.3 Calcul du mouvement

Afin de respecter un calcul issu du Principe Fondamental de la Dynamique, le mouvement lunaire est calculé à partir d'un modèle qui est facilement implémentable sur machine.

En effet c'est grâce à l'algorithme de Verlet à un pas que nous avons pu calculer différentes positions de la Lune à partir de la force (issue du PFD) et d'un temps infinitésimal séparant deux positions de la Lune.

Calcul du vecteur force (en deux temps)

où

— G représente la constante gravitationnelle
— d représente la distance entre la Terre et la Lune
— u et f le vecteur force à différents moment du calcul
— p la planète prise en paramètre (il s'agit de la Lune)

```
1    const double G = 6.67408e-11;
2    Vect u, f;  double d = u.norm();
3
4     u.setX(p.getPos().getX() - pos.getX());     u.setX(u.getX() / d);
5     u.setY(p.getPos().getY() - pos.getY());     u.setY(u.getY() / d);
6     u.setZ(p.getPos().getZ() - pos.getZ());     u.setZ(u.getZ() / d);
7
8     f.setX((G * getMass() * p.getMass()) / (d * d) * u.getX());
9     f.setY((G * getMass() * p.getMass()) / (d * d) * u.getY());
10    f.setZ((G * getMass() * p.getMass()) / (d * d) * u.getZ());
```

Voici la methode qui represente l'algorithme de verlet dans notre programme :

```
1  void Planet::movement(Planet p, float h, Vect f)
2  {
3  Vect a0(f.getX() / getMass(), f.getY() / getMass(), f.getZ() / getMass());
4
5    pos.setX(pos.getX() + getVel().getX() * h + 0.5 * (h * h) * a0.getX());
6    pos.setY(pos.getY() + getVel().getY() * h + 0.5 * (h * h) * a0.getY());
7    pos.setZ(pos.getZ() + getVel().getZ() * h + 0.5 * (h * h) * a0.getZ());
8
9      f = gForce(p);
10
11 Vect a1(f.getX() / getMass(), f.getY() / getMass(), f.getZ() / getMass());
12
13     vel.setX(vel.getX() + 0.5 * h * (a0.getX() + a1.getX()));
14     vel.setY(vel.getY() + 0.5 * h * (a0.getY() + a1.getY()));
15     vel.setZ(vel.getZ() + 0.5 * h * (a0.getZ() + a1.getZ()));
16 }
```

Chapitre 5

Conclusion

Ce projet a été enrichissant pour tous car il nous a permis de nous familiariser avec la bibliothèque OpenGL et d'apprendre son utilisation.

Il constitue une expérience supplémentaire quant à la gestion du travail collectif ; nous étions tour à tour désigné «leader» du groupe pendant deux semaines.

Enfin, il nous a permis d'appliquer les techniques de communication et de conduite de projet présentées dans une autre unité d'enseignement.

Remerciements

Nous tenons à remercier toutes les personnes ayant contribué au succès de ce projet.

Tout d'abord nous adressons nos remerciements à notre professeur encadrant, Mme Mountaz Hascoët qui nous à permis de faire nos premiers pas avec OpenGL et pour nous avoir accompagnés tout le long de ce projet.

Aussi, nous remercions vivement M. Jérome Dorignac professeur du département physique de la Faculté Des Sciences. En effet grâce à son aide, le mouvement lunaire adopte une trajectoire elliptique plutôt réaliste, contrairement aux versions antérieures.