

Système solaire

Rapport de projet

Réalisé par :

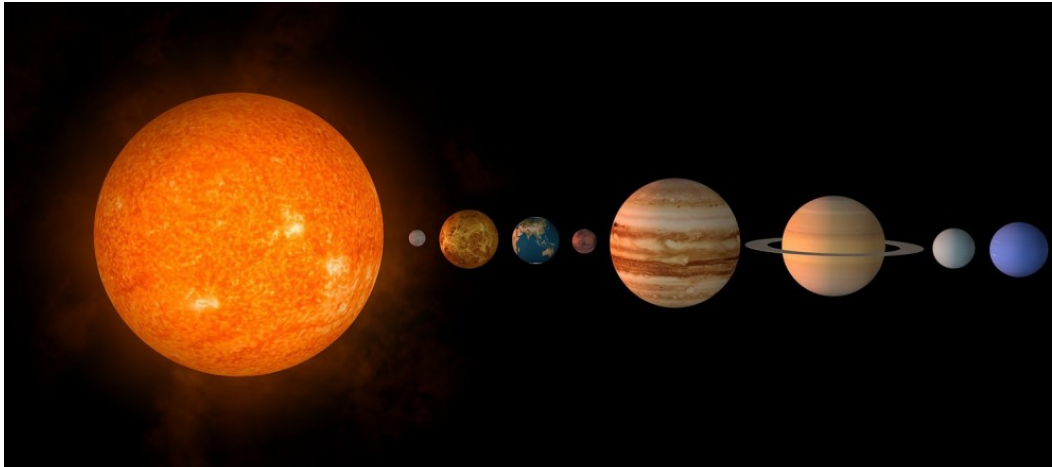
**BEN HAMOUDA Amel
HERBRETEAU Amélie**

Table des matières

I. Description du projet.....	3
II. Mode d'emploi.....	3
III. Diagramme des classes.....	5
IV. Description de l'architecture.....	5
1. Formes.....	5
2. Transformations.....	6
V. Planètes.....	7
VI. Conclusion.....	8
VII. Bibliographie.....	9
VII. Annexe.....	9
1. Instructions de compilation.....	9
2. Documentations / Instructions d'utilisation.....	9

I. Description du projet

Notre projet consiste à développer un système solaire composé d'une étoile, de 8 planètes et de 2 satellites :



Étoile	Soleil							
Satellites	Lune				Callisto			
Planètes	Mercure	Vénus	Terre	Mars	Jupiter	Saturne	Uranus	Neptune

Ce projet est développé en C++ à l'aide des bibliothèques OpenGL et C3GA.

Notre projet est principalement découpé en deux parties.

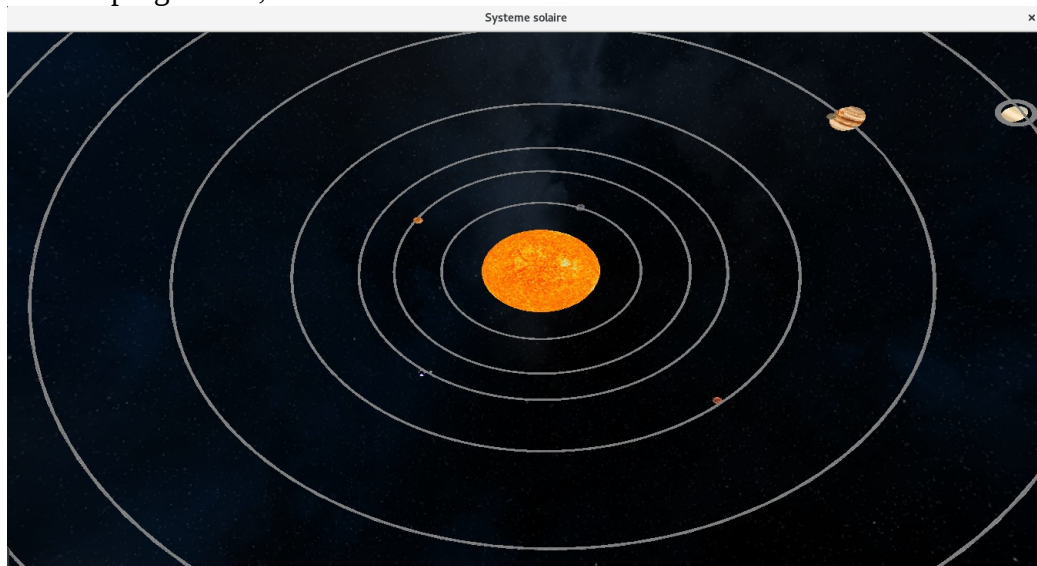
La première est la partie "transformation" dans laquelle se trouve tout les outils nous permettant d'effectuer des transformations sur les formes.

La seconde est la partie "formes" dans laquelle se trouve toute la partie nécessaire à la création des planètes et de l'anneau de Saturne.

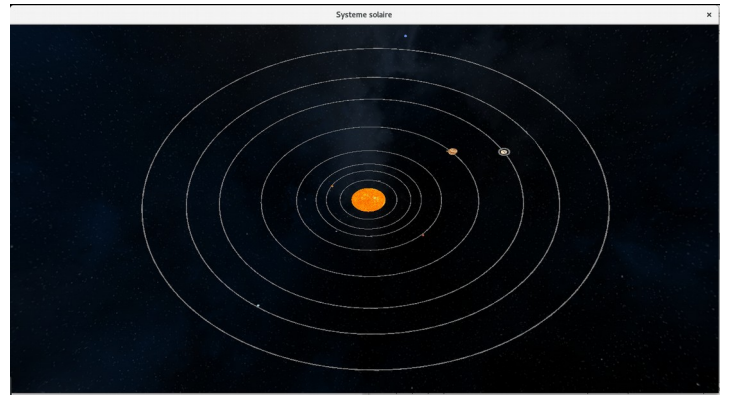
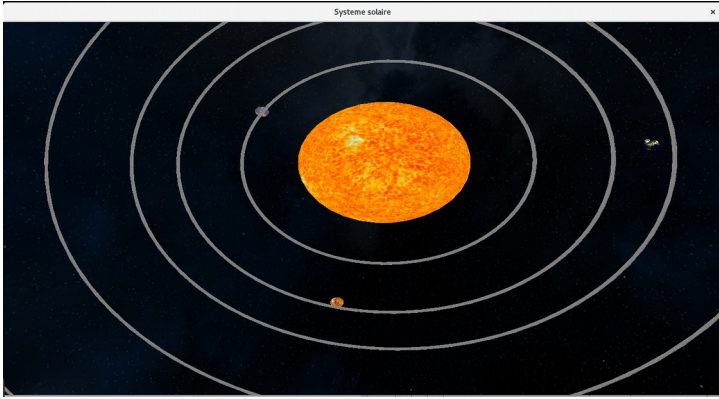
Ce rapport a pour but d'expliquer à l'utilisateur le mode de fonctionnement du projet et de décrire l'architecture que nous avons développé pour notre projet.

II. Mode d'emploi

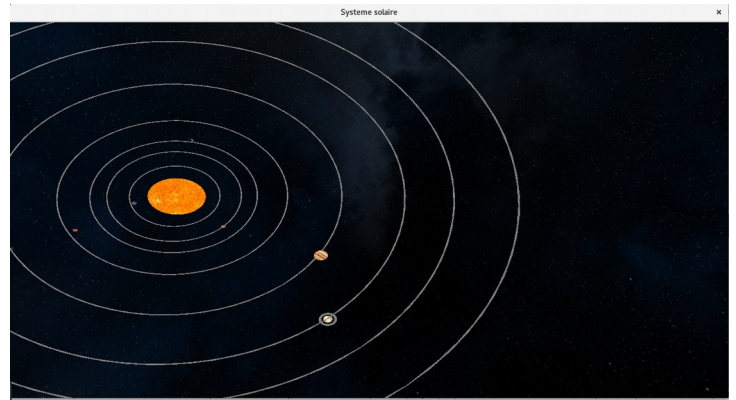
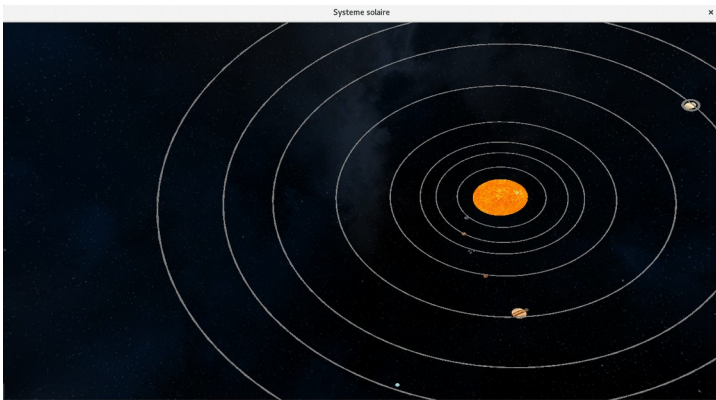
Au lancement du programme, on obtient la fenêtre suivante :



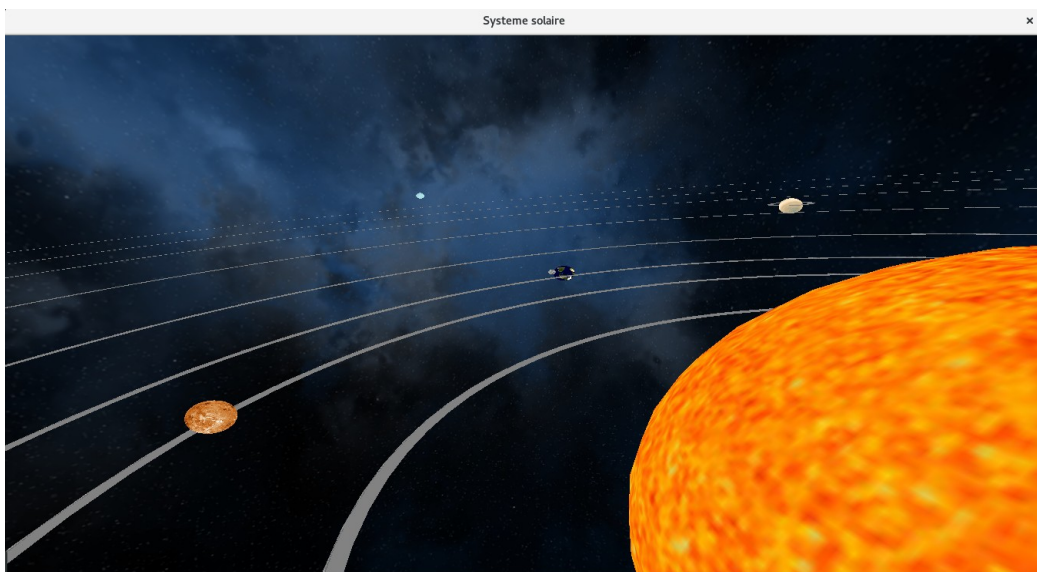
On peut zoomer avec la touche **z** et dé-zoomer avec la touche **s** du clavier :



On peut aller à gauche avec la touche **q** et à droite avec la touche **d** du clavier :

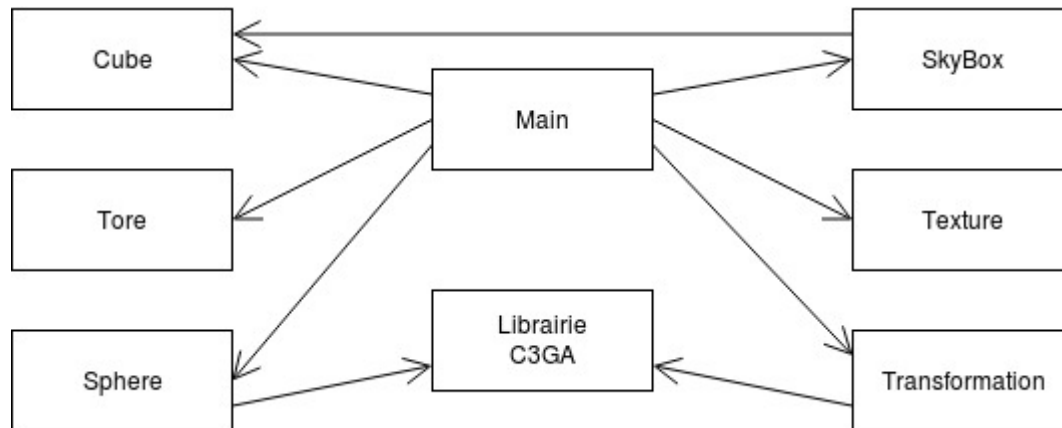


On peut changer d'angle de vue à l'aide de la souris en appuyant sur clique gauche + mouvement :



III. Diagramme des classes

Le schéma suivant montre l'inclusion de chaque classe et où on utilise la librairie C3GA :



La majorité des classes utilise aussi la librairie OpenGL.

IV. Description de l'architecture

1. Formes

Cette partie a été traitée en plusieurs classes, chaque classe représente une forme.

A – Cube

La classe Cube permet la création de cubes.

Elle est composée de plusieurs méthodes :

- un constructeur qui prend en paramètre la taille du côté du cube,
- une méthode "build" qui construit le cube en OpenGL à l'aide de la taille du cube,
- une méthode "gerDataPointer" qui renvoie les données des vertex du cube,
- une méthode "getVertexCount" qui renvoie le nombre de vertex.

Ainsi que de nombreux attributs :

- un vecteur de ShapeVertex pour stocker les vertices du cube en OpenGL,
- une variable qui stocke le nombre de vertex du cube ci-dessus.

Cette classe est utilisée pour la création de la skybox.

B – Sphère

La classe Sphère permet la création de sphères.

Elle est composée de plusieurs méthodes :

- un constructeur qui prend en paramètres le rayon, la latitude et la longitude,
- une méthode "sphere" qui crée une sphère en C3GA,
- une méthode "build" qui construit la sphère en OpenGL à l'aide du rayon calculé avec la méthode ci-dessus,
- une méthode "gerDataPointer" qui renvoie les données des vertex de la sphère,
- une méthode "getVertexCount" qui renvoie le nombre de vertex,

- une méthode "getSphere" qui renvoie la sphère C3GA,
- une méthode "setSphere" qui modifie la sphère C3GA.

Ainsi que de nombreux attributs :

- une sphère de type Mvec<double> pour stocker une sphère C3GA,
- une liste de coordonnées qui permet le stockage des coordonnées de la sphère ci-dessus,
- un vecteur de ShapeVertex pour stocker les vertices de la sphère en OpenGL,
- une variable qui stocke le nombre de vertex de la sphère ci-dessus,
- deux variables qui stockent la longitude et latitude de la sphère.

Cette classe est utilisée pour la création des planètes et du soleil.

C – Tore

La classe Tore permet la création de tores.

Elle est composée de plusieurs méthodes :

- un constructeur qui prend en paramètre le rayon interne et externe, et la valeur de chacun,
- une méthode "build" qui construit le tore en OpenGL à l'aide des paramètres du constructeur,
- une méthode "getDataPointer" qui renvoie les données des vertex du tore,
- une méthode "getVertexCount" qui renvoie le nombre de vertex.

Ainsi que de nombreux attributs :

- un vecteur de ShapeVertex pour stocker les vertices du cube en OpenGL,
- une variable qui stocke le nombre de vertex du cube ci-dessus.

Cette classe est utilisée pour la création des anneaux de Saturne et Uranus.

2. Transformations

Pour cette seconde partie on crée les méthodes pour chaque type de transformation :

- une méthode "translate" qui prend en paramètres le vecteur de la forme, le facteur de translation et l'axe de translation. Cette méthode calcule la translation d'un objet en C3GA.
- une méthode "rotate" qui prend en paramètres le vecteur de la forme, l'angle de rotation et l'axe de rotation. Cette méthode calcule la rotation d'un objet en C3GA.

→ une méthode "scale" qui prend en paramètres le vecteur de la forme et la valeur de scale. Cette méthode calcule le scale d'un objet en C3GA.

→ une méthode "applyTranslationX" qui prend en paramètre l'objet et applique la translation sur l'axe X de l'objet.

→ une méthode "applyRotation" qui prend en paramètre l'objet et applique la rotation.

→ une méthode "applyScale" qui prend en paramètre l'objet et applique le scale.

Les transformations ont été calculées à l'aide de la librairie C3GA en suivant ces formules :

→ Translation : $T \times O \times T^{-1}$

Avec $T = 1 - \frac{1}{2}t e_{\infty}$, où t est le vecteur de translation et O l'objet à traduire.


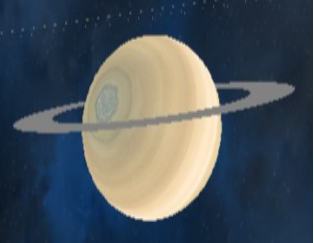



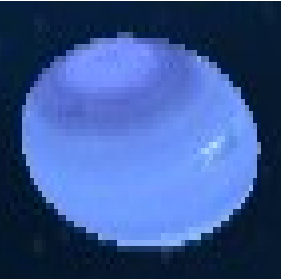
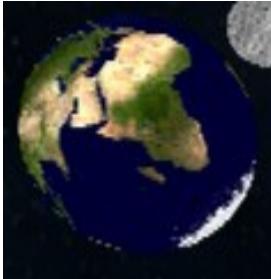

→ Rotation : $R \times O \times R^{-1}$


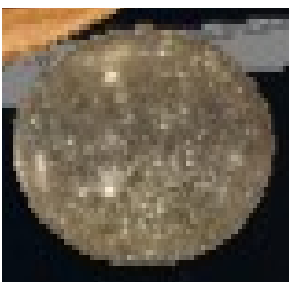

Avec $R = \cos(\frac{1}{2}\alpha) - B \sin(\frac{1}{2}\alpha)$, où α est l'angle de rotation, B le bi-vecteur associé à l'axe de rotation et O l'objet à rotater.

→ Scale : $D \times O \times D^{-1}$

Avec $D = 1 - \frac{1-s}{1+s}e_{o\infty}$, où s est le facteur de scale et O l'objet à scaler.

V. Planètes

Étoile Planètes Satellites	Images	Étoile Planètes Satellites	Images
Soleil		Saturne	
Mercure		Uranus	
Vénus		Neptune	
Terre		Lune	

Mars		Callisto	
Jupiter			

VI. Conclusion

Lors de ce projet nous avons pu mettre en application les connaissances liées à la programmation C++ et à l'algèbre géométrique. En développant un modèle de système solaire.

Le modèle obtenu dont les éléments principaux (notamment les planètes, et quelques satellites) ont été modélisés. On trouve aussi les trajectoires des différents astres afin de mieux suivre leur mouvement dans le modèle. Toutes les transformations ont été codées à l'aide de la librairie C3GA. La principale difficulté de ce projet a été le couplage entre la librairie de calcul d'algèbre géométrique (C3GA) et la librairie d'affichage 3D (OpenGL).

On pourrait penser (en tant que projet à plus long terme) à coder des librairies incorporant totalement l'algèbre géométrique à OpenGL, puisque OpenGL telle quelle n'est pas adaptée à recevoir des opérations et calculs effectués à l'aide de l'algèbre géométrique (on le voit rapidement lorsqu'on modélise des objets 3D de base).

VII. Bibliographie

→ Système solaire :

<http://www.planete-astronomie.com/>

http://sciences-physiques.ac-dijon.fr/archives/astronomie/systeme_solaire/tableau1.htm

→ Algèbre géométrique :

https://elearning.u-pem.fr/pluginfile.php/318286/mod_resource/content/1/cga.pdf

→ Librairie C3GA :

<https://github.com/vincentnozick/garamon>

VII. Annexe

1. Instructions de compilation

Pour compiler le programme avec les modules il suffit de lancer le CMakeList :

```
mkdir build
```

```
cd build
```

```
cmake ..
```

```
make
```

Pour l'exécution il suffit de taper :

```
./SystemeSolaire
```

2. Documentations / Instructions d'utilisation

Pour lancer le programme il suffit de taper *./SystemeSolaire* dans le terminal.

Ensuite une fenêtre s'ouvre, voir [Mode d'emploi](#) et le README.md pour la suite.