

---

# Système solaire

Projet de synthèse d'images

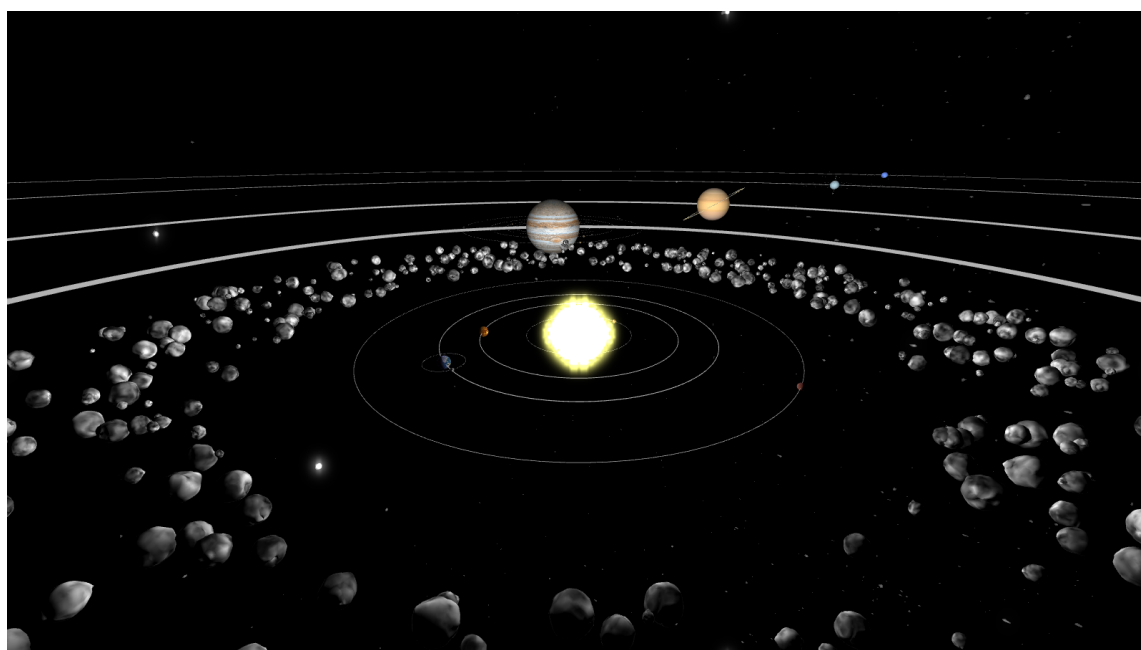
---

Kevin TRANCHO et Victor VEILLERETTE

Master Informatique 2<sup>ème</sup> année

Spécialité Sciences de l'image

Année 2018 - 2019

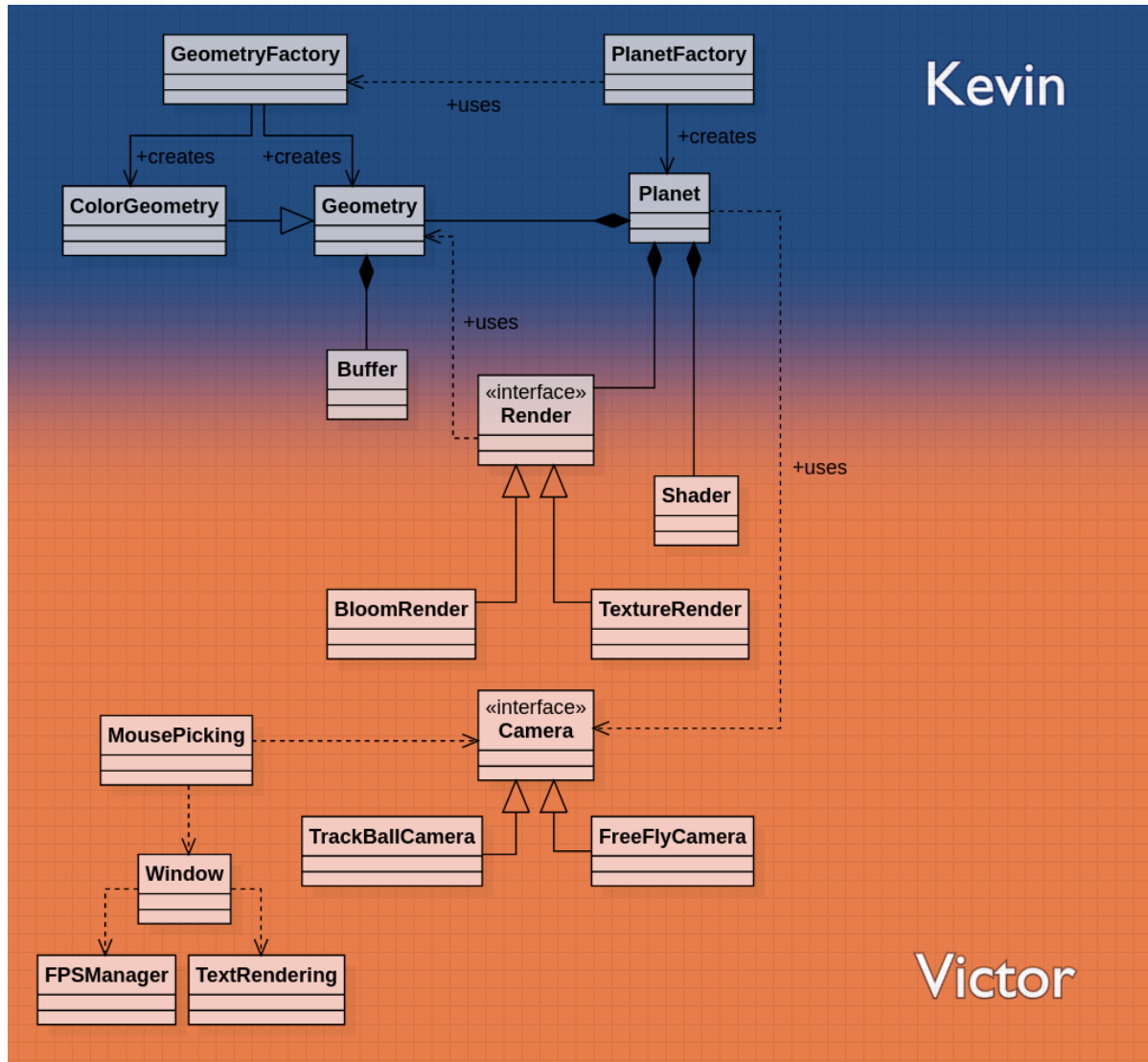


# Table des matières

<b>1</b>	<b>Architecture</b>	<b>2</b>
<b>2</b>	<b>Fonctionnalités basiques</b>	<b>2</b>
2.1	Surcouches à OpenGL et SDL . . . . .	2
2.2	Caméras . . . . .	3
2.3	Géométrie . . . . .	3
2.4	Planètes . . . . .	3
2.4.1	Planet . . . . .	3
2.4.2	PlanetFactory . . . . .	4
2.5	Astéroïdes . . . . .	4
2.5.1	Ico-sphère et génération . . . . .	4
2.5.2	Ceinture d'astéroïdes . . . . .	4
<b>3</b>	<b>Fonctionnalités supplémentaires</b>	<b>4</b>
3.1	Mouse picking . . . . .	4
3.2	Rendu de texte . . . . .	5
3.3	Anti-Aliasing MSAA . . . . .	5
3.4	Bloom Effect . . . . .	6
<b>4</b>	<b>Manuel</b>	<b>7</b>
4.1	Compilation et exécution . . . . .	7
4.2	Interface . . . . .	7
4.3	Fichiers de création de planètes . . . . .	8
4.3.1	Création . . . . .	8
4.3.2	Attributs Planète . . . . .	9
4.3.3	Attributs Ceinture d'astéroïdes . . . . .	9

# 1 Architecture

L'architecture que nous proposons est la suivante :



Dans la séparation des tâches, Kevin était principalement chargé de gérer que qui touche à la géométrie et gestion des planètes et Victor avait pour rôle principal de s'occuper de la partie qui s'intéresse au rendu, vue depuis une caméra et gestion des shaders.

## 2 Fonctionnalités basiques

### 2.1 Surcouches à OpenGL et SDL

Plusieurs classes permettent de gérer plus facilement les fonctions OpenGL/SDL de base niveaux :

- La classe **Buffer** permet de gérer les différents VBO, VAO, IBO. On lui fournit un ensemble de sommets (VERTEX, NORMAL, TEXTURECOORD) et de faces et il se charge automatiquement de générer les buffers en mémoire GPU et à les remplir correctement. Cela

permet de simplifier grandement la création des géométries.

- La classe **TextureRender** permet de charger une image à partir d'un path et de correctement remplir les textures du GPU et d'envoyer automatiquement les textures au bon endroit dans les shaders.
- La classe **Shader** permet l'utilisation simplifiée des shaders en retrouvant automatiquement l'emplacement des variables uniformes grâce à une map :

```
void uniformMatrix(std::string key, glm::mat4 & mat);  
void uniformValue(std::string key, GLfloat value);  
void uniformVector(std::string key, glm::vec4 vec);
```

- Enfin, les classes **Window** et **FPSManager** englobent les fonctions SDL et permettent :
  - D'ouvrir/fermer une fenêtre
  - De gérer le clavier
  - De lancer la boucle principale de l'application en utilisant deux lambdas : une pour la gestion des événements et l'autre pour le rendu. Cette boucle s'assure en interne que le nombre de FPS soit constant (option réglable) :

```
void run(std::function<bool(SDL_Event *)> play, std::function<void()> draw);
```

## 2.2 Caméras

L'interface **Camera** permet de généraliser le principe et de rendre facile le changement de caméra pendant l'exécution. Elle demande l'implémentation suivante :

```
virtual glm::mat4 getViewMatrix() const = 0;  
virtual glm::vec4 getPosition() const = 0;  
virtual void gestEvent(SDL_Event *event) = 0;
```

Deux caméras ont été implémentées : la **TrackBall Camera** qui permet de tourner autour d'un objet (planètes, satellites, astéroïdes) ainsi que la **Freefly Camera** qui permet de se déplacer dans toutes les directions à la manière d'une caméra FPS.

## 2.3 Géométrie

Nous proposons un module **Geometry** permettant de garder en mémoire une forme générée : sommets et faces. La création des différentes primitives classiques est proposée à l'aide de **GeometryFactory**. Celle-ci permet principalement de générer les UV-sphères pour les planètes, les tores pour les trajectoires, la forme d'une Skybox pour le ciel étoilé, mais aussi les astéroïdes.

## 2.4 Planètes

### 2.4.1 Planet

La classe **Planet** fait le lien entre la géométrie et le rendu. C'est-à-dire qu'elle connaît une unique géométrie à laquelle on peut ajouter des textures ou d'autres composants de rendu tels que des anneaux, une atmosphère par exemple. L'autre idée du module Planet est d'être générique et permettre la récursivité. C'est-à-dire qu'une planète peut aussi bien être un soleil, un satellite ou un astéroïde.

### 2.4.2 PlanetFactory

La classe **PlanetFactory** a pour fonction principale de lire la recette de fabrication d'une planète, soleil, ceinture d'astéroïdes depuis un fichier : le système solaire proposé est chargé depuis `assets/solar_system`.

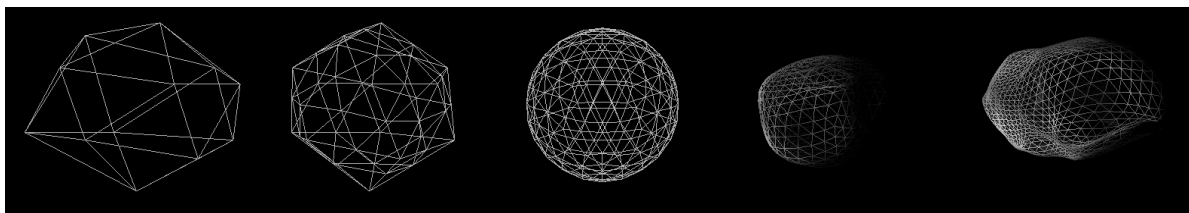
## 2.5 Astéroïdes

### 2.5.1 Ico-sphère et génération

Nous avons choisi de générer les astéroïdes, l'idée était de le faire depuis une sphère. Mais les UV-sphères ne permettant pas une déformation homogène de la surface (à cause des pôles). Nous proposons l'**Ico-sphère**. À noter que l'Ico-sphère est aussi utilisée pour l'atmosphère des planètes et donne de meilleurs résultats qu'une UV-sphère.

Pour construire une Ico-sphère nous construisons un géodésique simple (ici les sommets sont construits à partir de 3 rectangles orthogonaux). L'étape suivante est de le subdiviser puis projeter chaque sommet sur une sphère (renormalisation à distance 1 de l'origine). Pour fabriquer un astéroïde, l'idée est de générer aléatoirement des potentiels (ici sources avec une force positive ou négative), puis de calculer la somme des déformations de chaque sommet vers un potentiel donné selon un coefficient de déformation fixé. L'augmenter le nombre de potentiels va augmenter le nombre de cabossages de la forme obtenue et augmenter le coefficient de déformation va augmenter les étirements de la forme mais aussi les irrégularités.

Sur la figure suivante nous illustrons premièrement le géodésique, puis une subdivision de celui-ci. Ensuite sont affichés une ico-sphère et deux astéroïdes (un cabossé par 200 potentiels pour une déformation de 10 et l'autre 500 potentiels et déformation 20).



### 2.5.2 Ceinture d'astéroïdes

La ceinture d'astéroïdes est obtenue par génération aléatoire de tailles, rotation, positions aléatoire pour des valeurs dans des intervalles donnés pour un ensemble d'astéroïdes générés dans une zone définie par un anneau. Ici si on demande  $N$  astéroïdes, on générera de l'ordre de  $3 \log_2(N)$  géométries différentes d'astéroïdes.

## 3 Fonctionnalités supplémentaires

### 3.1 Mouse picking

On souhaite qu'au clic de la souris sur une planète, notre caméra se déplace vers celui-ci. Pour cela, on va lancer un rayon de la caméra passant par le point du clic et on cherche alors à voir si il intersecte un des objets de notre scène.

C'est le rôle de la classe **MousePicking**. Elle utilise l'algorithme suivant :

1. Transformations en coordonnées homogènes clippées  $([-1;1], [-1;1], -1, 1)$  :

```
vec4 coords = ((2 * x) / W - 1, (1 - 2 * y / H))
```

2. Transformations en coordonnées de l'espace view :

```
vec4 coordsView = vec4(vec2(inverse(projectionMatrix) * coords), -1, 0)
```

3. Transformations en coordonnées de l'espace monde :

```
vec3 final = inverse(ViewMatrix) * coordsView
```

Il est maintenant possible de vérifier l'intersection de notre rayon (qui part de la caméra) avec les objets de notre scène pour déterminer si on clique sur un élément. Ici on considèrera uniquement le plus proche objet qui intersecte.

### 3.2 Rendu de texte

On a souhaité faire un mécanisme de rendu de texte simple pour permettre d'afficher les FPS à l'écran par exemple.

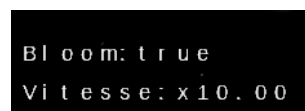
Nous avons choisi la technique des **bitmap font**. On a donc généré la texture suivante à partir de la police Arial :



Lors du rendu, on dessine un rectangle à l'écran pour chaque lettre à écrire et on utilise les bonnes coordonnées de notre texture en fonction de la lettre à dessiner :

```
float ux = (car % 16) / 16
float uy = (car / 16) / 16
```

On rend ensuite la couleur rouge transparente dans notre shader pour finaliser le dessin. On obtient par exemple :



### 3.3 Anti-Aliasing MSAA

Nous nous sommes vite confrontés aux problèmes de crénelage mais surtout de la visualisation lointaine des petits objets (trajectoires).

Pour cela, nous avons opté pour le multisampling (MSAA) à 4 points. Pour cela, on crée un framebuffer que l'on associe à une texture `GL_TEXTURE_2D_MULTISAMPLE`. On effectue le rendu grâce à un multisample renderbuffer associé.

Pour finir, on résoud le problème de la sélection en passant par un deuxième framebuffer qui sélectionne le point le plus proche : (`GL_NEAREST`).

On observe une baisse de performance d'environ 25% pour une résolution 1920/1080, et une baisse de 35% pour du MSAA 8x avec la même résolution.

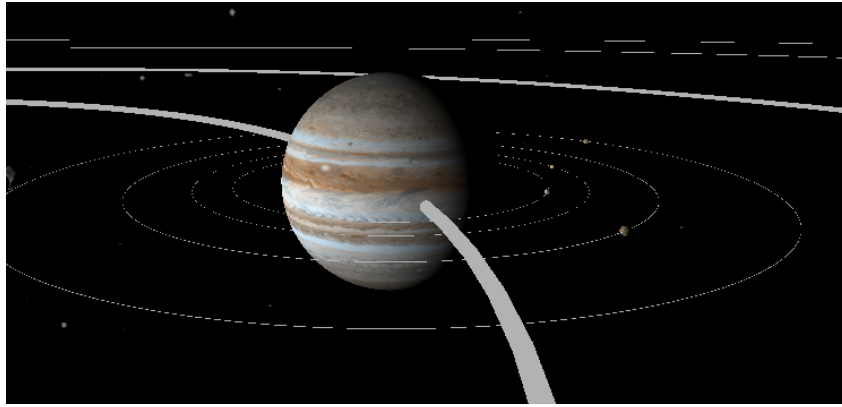


FIGURE 1 – Problème d'aliasing et de visualisation des objets fins

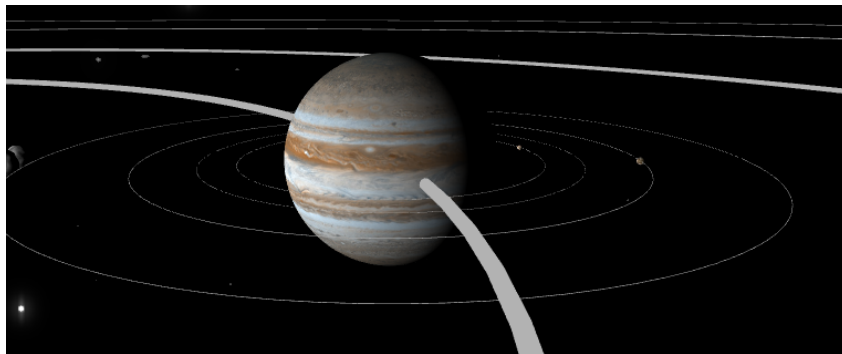


FIGURE 2 – On a bien amélioré la qualité de rendu avec du MSAA 4x

### 3.4 Bloom Effect

On s'est également aperçu que le rendu du soleil ne donnait vraiment pas l'impression d'une source lumineuse :

L'idée de l'algorithme est alors le suivant :

1. On effectue un rendu de notre scène standard avec deux sorties : la couleur normale et la couleur bright qui devra s'illuminer :

```
layout (location = 0) out vec3 fFragColor;
layout (location = 1) out vec3 fBrightColor;
```

2. On effectue plusieurs itérations de flou gaussien (en deux étapes) sur la texture bright de notre scène.
3. On combine les deux pour obtenir le rendu final

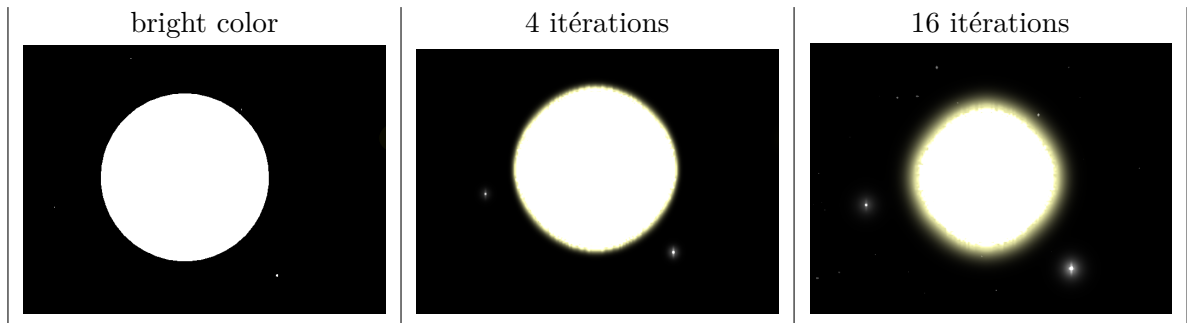


FIGURE 3 – Evolution de l'algorithme avec 4 et 16 itérations

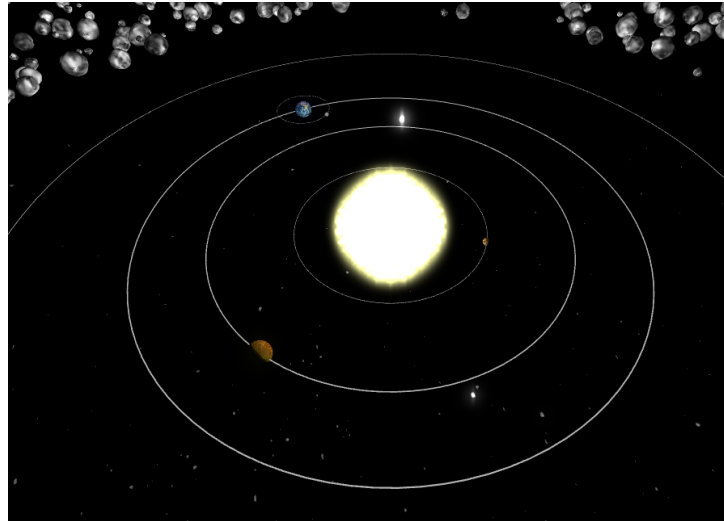


FIGURE 4 – Le rendu avec bloom et MSAA est beaucoup plus satisfaisant.

## 4 Manuel

### 4.1 Compilation et exécution

Pour compiler l'application :

```
mkdir build
cd build
cmake ..
make
```

Pour lancer l'application :

```
./app/main
```

Si besoin, utiliser le mode de compatibilité mesa :

```
MESA_GL_VERSION_OVERRIDE=3.3 ./app/main
```

### 4.2 Interface

Le système solaire est alors chargé et affiché à l'écran. On peut se déplacer grâce à la souris et au clavier. Pour déplacer l'angle de vue, il faut laisser enfoncé le clic gauche et bouger la souris.

Le tableau suivant résume les touches utiles :



Touche	Description
ESC	Sortir de l'application
TAB	Afficher les meshes
C	Changer de caméra entre TRACKBALL et FREEFLY
B	Activer ou Désactiver le bloom et le MSAA
SPACE	Arrêter/Relancer le temps
Flèche gauche	Ralentir le temps
Flèche droite	Accélérer le temps
Souris + ZQSD	Déplacer la caméra
Clic sur un objet	Déplacer la caméra vers cet objet

FIGURE 5 – Manuel d'utilisation

## 4.3 Fichiers de création de planètes

### 4.3.1 Création

Pour démarrer une création, il y a deux possibilités : créer l'entité ou la charger depuis un fichier externe. Il est possible de créer un satellite récursivement avec la même syntaxe en donnant simplement l'attribut satellite.

Pour charger depuis un fichier externe :

```
load [PATH]
```

ou générer la planète avec une syntaxe de la forme :

```
create [TYPE]
  [ATTRIBUT] [VALUES]
  [ATTRIBUT] [VALUES]
  ...
  [ATTRIBUT] [VALUES]
satellite create [TYPE]
  [SATELLITE-ATTRIBUT] [VALUES]
  ...
end
...
satellite load [PATH]
...
end
```

Les différentes valeurs pour type sont les suivantes :

Type	Description
Planet	création d'une planète
Sun	création d'un soleil
Asteroid-ring	génération d'une ceinture d'astéroïdes
Obj	suivi du chemin du .obj à charger

À noter que les attributs donnés pour Planet sont aussi valables pour Sun et Obj.

#### 4.3.2 Attributs Planète

```
create [Planet | Sun | Obj [PATH]]
  diameter [FLOAT: diamètre de la planète en km]
  parent-distance [FLOAT: distance au parent en km]
  orbital-period [FLOAT: période de rotation autour du parent en jours]
  rotation-period [FLOAT: période de rotation sur elle-même en heures]
  texture [PATH: chemin de la texture]
  texture-day [PATH: chemin de la texture principale]
  texture-night [PATH: chemin de la texture vue sur l'ombre]
  clouds [PATH: chemin de la texture en rotation en surface]
  ring [FLOAT: diamètre de l'anneau]
  color [FLOAT: couleur unie de la planète]
  ringtexture [PATH: chemin de la texture de l'anneau]
  ringalpha [PATH: chemin de la texture pour le canal alpha de la texture]
  ringangle [FLOAT: angle d'inclinaison de l'anneau]
  atmosphere [FLOAT: taille de l'atmosphère]
  atmosphereColor [vec3: couleur de l'atmosphère]
  satellite [GENERATION COMMAND: recette de fabrication du satellite]
end
```

#### 4.3.3 Attributs Ceinture d'astéroïdes

```
create Asteroid-ring
  diameter [FLOAT: diamètre interne en km] [FLOAT: diamètre externe en km]
  orbital-period [FLOAT: période de rotation autour du parent en jours]
  asteroid-period [FLOAT: période minimale en heures] [FLOAT: période maximale]
  (période aléatoire de rotation des astéroïdes sur eux-mêmes en heures)
  asteroid-diameter [FLOAT: diamètre minimum en km] [FLOAT: diamètre maximum]
  (diamètre aléatoire des astéroïdes)
  asteroid-count [INT: nombre d'astéroïdes à afficher]
  resolution [INT: nombre de subdivision des astéroïdes, conseillé entre 3 et 5]
  cabossages [INT: nombre de cabossage des astéroïdes]
  deformation [FLOAT: coefficient de déformation]
end
```