ANIM3D - Super Mario Galaxy Remake

Maxime Legros - Melwan Chevassus

28 Novembre 2025

1 Introduction

L'idée du projet est de recoder une version minimale du jeu *Super Mario Galaxy*, sorti pour la première fois sur *Wii* en 2007. Le projet est donc composé d'une scène 3D sur laquelle apparaissent notamment des planètes, un joueur et des trous noirs, tout ceci dans l'espace.

Nous sommes repartis du TP 4 - **Shape matching** afin de pouvoir modifier l'attraction des objets.

2 Instructions de build

La soumission sur Moodle vient avec deux binaires préconstruits (ils doivent être lancés sur une architecture Linux x86 64 bits). Ces deux binaires sont deux versions du projet qui sont encore incompatibles entre elles :

- la version `camera_movement` fixe la caméra à proximité du joueur,
- la version `black_hole` laisse la caméra flottante dans la scène (ce qui permet notamment de jeter des objets dans un trou noir facilement).

Le build par défaut, exécuté avec la commande

```
cmake -B build && cmake --build build
```

donne la version `camera_movement`.

3 Gravité et animation

3.1 Planètes

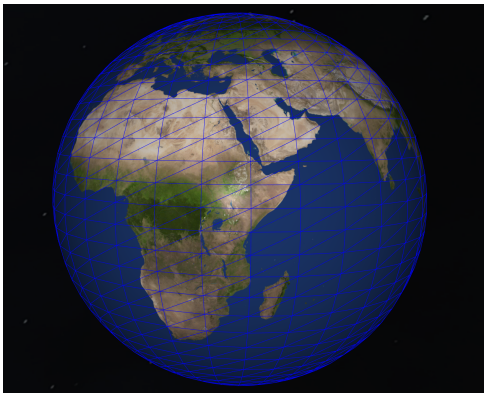
3.1.1 Affichage

Le premier élément à intégrer au projet est la gravité par planète. Chaque planète possède un rayon (qui est le rayon du mesh sphérique), et un rayon d'attraction plus grand que le rayon de la planète.

La texture des planètes est un planisphère de la terre, le même utilisé sur toutes les planètes et est tirée de ce site.

L'affichage de la planète ressemble à ceci :

(a) Planet texture, no wireframe



(b) Planet texture, with wireframe

L'affichage des planètes est géré dans la fonction `display_frame` de la scène.

3.1.2 Calcul de la gravité

Tous les objets qui sont à l'intérieur du rayon d'attraction d'une planète sont attirés dans la direction de cette planète. Pour éviter des conflits de gravité, on suppose que les planètes sont **toujours suffisamment éloignées** pour qu'un objet soit attiré par une planète au maximum.

La formule de gravité, telle qu'on la connaît dans la physique, est

$$\vec{F} = G\frac{m_1 m_2}{d^2}\vec{u}$$

où G est la constante de gravitation, m_1 la masse de la planète, m_2 la masse de l'objet, d la distance entre les deux centres de gravité et $\vec{u}$ le vecteur unitaire dirigé de l'objet vers la planète. Par souci de simplification, la gravité est implémentée avec la formule

$$\vec{F} = \frac{c}{d^2}\vec{u}$$

où c est une constante numérique.

La gravité des planètes est calculée dans la fonction `planet_attraction` du fichier `simulation.cpp`.

3.2 Trous noirs

3.2.1 Affichage

Comme dans le vrai jeu, les trous noirs sont affichés comme un **billboard** en deux dimensions : la texture est simplifiée et est toujours face au joueur. Le billboard de trou noir subit donc une rotation en fonction de la position de la caméra par rapport au trou noir. Ainsi, le trou noir est affiché en utilisant un `mesh_primitive_quadrangle`.

La texture du trou noir, tirée du site Mario Wiki est en partie transparente, les trous noirs sont donc dessinés en dernier à l'intérieur de la fonction `display_frame`.

Voici un exemple d'affichage lorsqu'un trou noir :

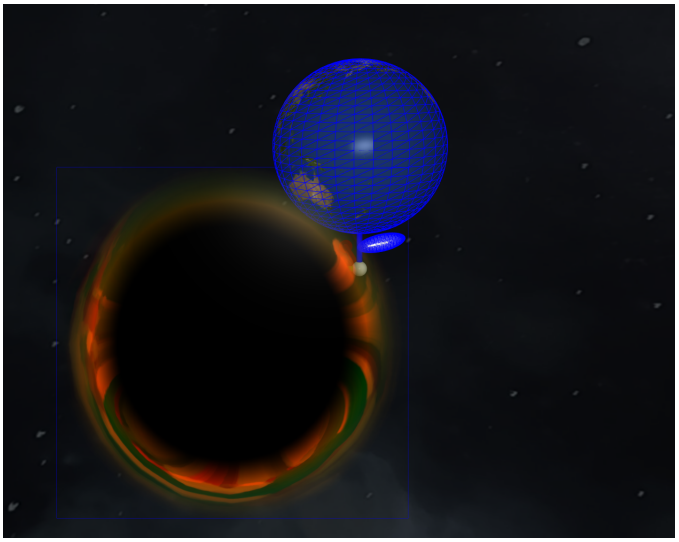


Figure 2: Black Hole display (with wireframes)

La gravité des trous noirs est calculée dans la fonction `simulation_step` du fichier `simulation.cpp`.

3.2.2 Calcul de la gravité

Le calcul de la gravité du trou noir est sensiblement le même que celui pour les planètes, seulement la constante c est plus élevée ici.

Lorsqu'un objet est emprisonné dans le rayon d'attraction d'un trou noir, il lui est impossible d'en sortir. L'animation du trou noir (à durée réglable) se termine par la suppression du déformable.

Nous avons tenté de reproduire l'animation du jeu pour les trous noirs avec une trajectoire en spirale dirigée vers le trou noir via les formules :

- Calcul de la nouvelle taille de l'objet aspiré : pour chaque sommet k de l'objet,

$$\text{obj}_{\text{pos}_k} = T(\text{obj}_{\text{com}})S(1 - 2 * \text{dt})T(-\text{obj}_{\text{com}}).$$

  où $T(x)$ est la matrice de translation du vecteur x, $S(t)$ est la matrice de scale de paramètre $t > 0$ et dt est la durée de la frame.

- Calcul de la nouvelle vitesse de l'objet aspiré : pour chaque sommet k de l'objet,

$$\text{obj}_{\text{vit}_k} = (\text{hole}_{\text{pos}} - \text{obj}_{\text{com}}) \times (\text{cam}_{\text{pos}} - \text{obj}_{\text{pos}_k})/\text{dt}.$$

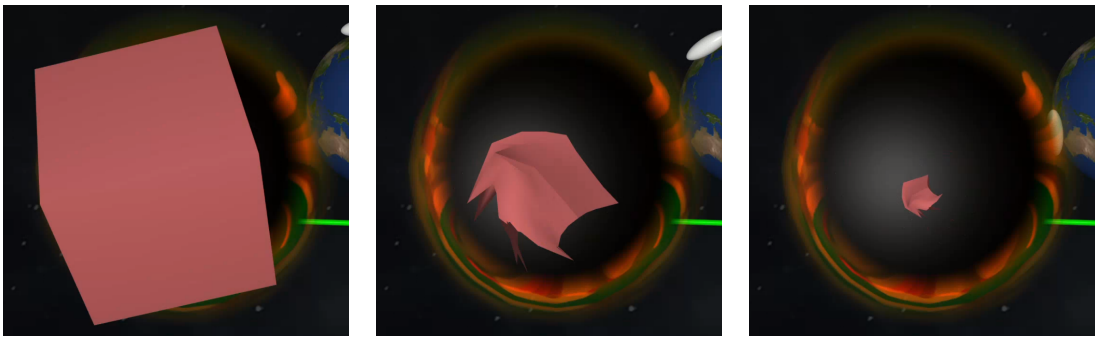
- Calcul de la nouvelle position de l'objet aspiré : pour chaque sommet k de l'objet,

$$\text{obj}_{\text{pos}_k} = c_1 \text{normalize}(\text{obj}_{\text{vit}_k}) + c_2(\text{hole}_{\text{pos}} - \text{obj}_{\text{com}}).$$

- Lorsque l'animation a duré suffisamment longtemps (en général après 4 ou 5 tours de spirale), l'objet disparaît de la scène.

3.3 Spaghettification

Le calcul d'attraction du trou noir permet l'affichage d'un effet de **spaghettification** (déformation plastique) :



3.4 Lancement et suppression de déformables

Le lancement de déformables à l'aide de la touche SPACE a été conservé pour interagir avec le joueur via les collisions. La touche K a été bind sur la suppression de tous les déformables, y compris le joueur.

4 Skybox

Un dernier élément de décoration a été intégré : la **skybox**. C'est en réalité un simple mesh sphérique bindé à la texture souhaitée.

La skybox est créée à l'initialisation de la scène, et bougée avec la caméra de sorte que la sphère soit centrée sur la caméra avec un rayon suffisant pour donner l'impression d'espace. Pour une raison inconnue, la tessellation de la skybox est déterminante sur la fréquence d'image. Afin de garder une certaine fluidité, la tessellation est gardée en `(16, 16)`. La texture n'étant pas répétée, des artéfacts apparaissent sur les pôles.

5 Placement de la caméra

Il existe deux types de caméra dans le projet :

- La caméra flottante, telle qu'elle est créée pour le TP,
- La caméra fixée sur le joueur, visible avec le binaire `camera_movement` (ou avec la compilation classique). Cette caméra est paramétrée par la distance au joueur ainsi que la normale de la planète. Lorsque le joueur quitte l'orbite de la planète, la caméra reste fixée à la planète. Dans ce mode, les bindings de `GLFW` concernant les mouvements de caméra ne fonctionnent pas. La position de la caméra est obtenue par

$$\text{cam}_{\text{pos}} = \text{average}(\text{obj}_{\text{pos}}) + \text{normalize}(\text{average}(\text{obj}_{\text{pos}}) - \text{planet}_{\text{center}}) * \text{cam}_{\text{dist}}$$

6 Configuration globale de la scène

La scène est parsée depuis des fichiers YAML selon cette architecture :

```
config
|-- black_holes
|   `-- black_hole_01.yaml
|-- camera
|   |-- camera_01.yaml
|   |-- camera_02.yaml
|   `-- camera_03.yaml
|-- planets
|   |-- planet_01.yaml
|   `-- planet_02.yaml
`-- scenes
    |-- scene_01.yaml
    `-- scene_02.yaml
```

Le fichier principal est celui de la scène (par exemple, `scenes/scene_02.yaml`), il importe les fichiers de configuration pour les planètes et pour les trous noirs en utilisant leur identifiant. Par exemple, ces lignes :

```yaml
planets:
    - 1
    - 2
```

dans le fichier de scène importe les planètes ayant pour identifiants 1 et 2 dans la scène. Ces planètes ont leur configuration dans les fichiers `planet_01.yaml` et `planet_02.yaml`.

Les valeurs inscrites dans les fichiers `.yaml` peuvent être modifiées par l'utilisateur au runtime, ce qui évite une nouvelle compilation.

L'identifiant de la scène à charger peut être modifié via le premier argument du binaire (seulement dans le binaire `camera_movement`).

7 Conclusion

Bien que la partie artistique ne soit pas la même que dans le vrai jeu, la nouvelle mécanique d'attraction gravitationnelle qui distingue ce jeu des autres jeux de platformer 2D/3D a bien été implémentée, et le rendu nous semble satisfaisant.