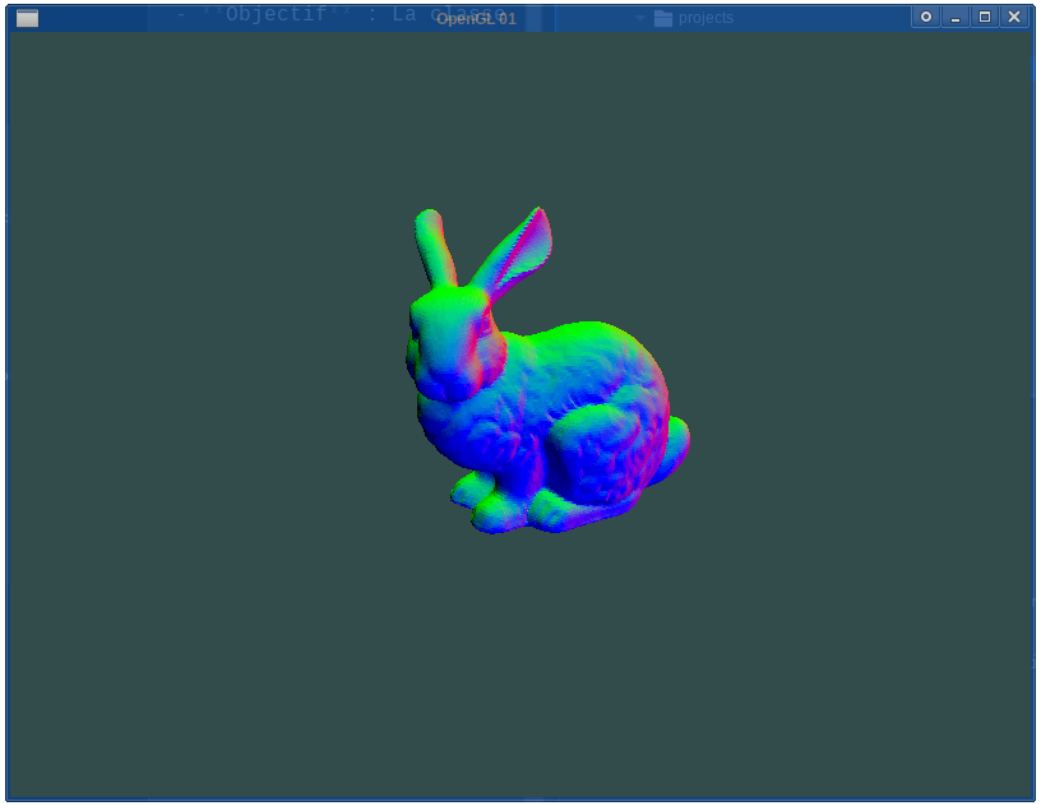
TP2 Space-Shooter 2D



Jehan-Antoine Vayssade

Prise en main

- **Donnée** : copier les dossiers, `w64devkit` et `opengl-cpp-template-tp2`
- **Compilateur** : w64devkit -> portable
- **Terminal** : lancé `w64devkit.exe`
- **Compilation** : `cd dossier_tp ; make -j 4`
- **Éxécutable** : `bin/mygame.exe`
- **Debug** : `gdb bin/mygame.exe`

Étapes

- Ajouter des asteroids a des positions aléatoires
  - Respecter la @boundary
  - Relancer un jet aléatoire tant qu'il y a overlapping
- Simuler le déplacement des astéroides (volocité+direction+dt xy)
  - Définir l'orientation des astéroides
  - Définir le déplacement du vaiseau
- Détecter les collisions et rebonds
  - D'abbord le cas avec la @boundary de l'univers
  - Puis entre chaque éléments de la scene
- Finalement lancé des particles pour détruire les astéroides
  - Gérer les intéractions entres les différents types d'objets
- Si vous avez le temp, faire un HUD
- Bonus vérifier la stabilité numérique (pensé a clamp certaines valeurs)

Architecture du moteur 3D

Composants principaux

- Device : Gestion du matériel, de la fenetre et des entrées utilisateur
- Driver : Abstraction de l'API graphique (OpenGL)
- GUI : Interface utilisateur
  - A regarder ?
- IO : Gestion des system de fichier (std, virtual, zip)
- Loader : Chargement des ressources (modèles, textures)
- Math : Classes mathématiques (vecteurs, matrices, etc.)
  - KDTree voir
- Node : Graphe de scène et gestion des objets 3D
  - Caméra

Architecture du jeux

- Core : Crée la fenetre et les instances partagers (driver, io, loader, ...) et les états potentiels du jeux
  - Menu principal, lancé une partie, settings, ...
  - Gère partielement la transition entre les états
- PostProcessing : gère les effets spéciaux (pas le temps)
- SpaceShooterState -> Game : Éxecute les logiques du jeux, les rendues, etc
- Object : Élement tangible du jeux (simulation physique) -> Astéroide
- Ship : Le vaiseau, qui est un Object
- Ammo : Idem, une particule qui représente un tir du vaiseau

Ajouter des asteroids

Voir le code dans `src/space-shooter/state/Game.cpp`

```
void Game::spawnAsteroid(int i, int k)
    std::mt19937 gen;
    std::uniform_real_distribution<float>
```

Implémenter la fonction isColliding dans `src/space-shooter/nodes/Object.cpp`

Soit deux objets A et B, avec :

- p_A et p_B leurs positions respectives
- r_A et r_B leurs rayons respectifs
- m une marge supplémentaire
- $\vec{r} = p_B - p_A$
- $d = |\vec{r}|$ (distance entre les centres des objets)

Trouver la condition de collision !

Simuler le déplacement des astéroides

Voir le code dans `src/space-shooter/node/Object.cpp`

Équations de mouvement

- force : $\vec{F} \in \mathbb{R}^2$
- mass : $m \in \mathbb{R}$
- acceleration : $\vec{a} = \frac{\vec{F}}{m}$
- velocity : $\vec{v}_{t+1} = \vec{v}_t + \vec{a}\Delta t$
- position : $\vec{p}_{t+1} = \vec{p}_t + \vec{v}_{t+1}\Delta t$

Fonction `Object::update` a compléter

Définir la rotation de l’astéroide

- position = $\vec{p}_t$
- $\vec{v} = \vec{p}_{t+1} - \vec{p}_t$
- Si $\|\vec{v}\| < 10^{-4}$:
  - $\text{rotation} = (0, 0, 0)$
- Sinon :
  - $\theta = -\arctan 2(v_y, v_x)$
  - $\theta_{\text{degrés}} = \theta \cdot \frac{180}{\pi} - 90$

Définir le déplacement du vaiseau

Jouer avec la direction pour influencer la vélocité !

Détecter les collisions et rebonds

Dans la dernière boucle de `Game::simulate`

1. Vérifier si la position de l'objet sort de l'univers

- si oui, repositionner l'objet, modifier la velociter en concequence

Dans `Game::handleCollision`

2. Définition des variables :

- $\vec{v_1}, \vec{v_2} = \text{vitesse initiale de l'objet 1 et 2}$
- $m_1, m_2 = \text{masse de l'objet 1 et 2}$

2. Calcul des nouvelles vitesses après collision :

- $\vec{v_1'} = \frac{\vec{v_1}(m_1 - m_2) + 2m_2\vec{v_2}}{m_1 + m_2}$
- $\vec{v_2'} = \frac{\vec{v_2}(m_2 - m_1) + 2m_1\vec{v_1}}{m_1 + m_2}$

3. Limitation des vitesses (fonction clamp) :

- Pour éviter des instabilités numérique, pensser a limité vos vitesse

Calcule des force d’attraction

Fonction `Game::calculateForce`

$$\vec{r} = \vec{p_2} - \vec{p_1}$$
$$|\vec{r}| = \sqrt{r_x^2 + r_y^2}$$
$$\hat{r} = \frac{\vec{r}}{|\vec{r}|}$$
$$\vec{F} = \frac{G m_1 m_2}{|\vec{r}|^2} \hat{r}$$

Où :

- $\vec{p_1}, \vec{p_2}$ sont les positions des objets
- m_1, m_2 sont les masses des objets
- G est la constante gravitationnelle, on prendra 0.01

Lancé des particles pour détruire les astéroides

- complété la fonction `Ship::update` qui ce charge d'instancier les particules

A vos clavier !