Rapport de projet INF443
Super Mario 64: Bob-Omb Battlefield

Maxime Peim
Alexandre Thiault

1 Introduction

Le projet sur lequel nous avons travaillé consiste en une reproduction partielle du premier niveau du jeu vidéo *Super Mario 64*, conçu au milieu des années 1990, pionnier du jeu vidéo 3D chez Nintendo et record de ventes pour l'époque. La partie du niveau *Bob-omb Battlefield* que nous avons recréée consiste en une scène fixe inanimée (décor muni de collisions : sols, murs...), des objets "non vivants" dotés d'une certaine animation (arbres, pièces rouges et jaunes, étoiles, pont qui se balance, bulle d'eau), et des personnages dotés de comportements qui les interconnectent (Chomp, Bob-ombs noires, Mario volant) dont un est contrôlable au clavier, le Bob-omb rose, pour qui nous avons rendu possible le suivi par la caméra.

2 Eléments de la scène 3D

2.1 Chomp

Le Chomp est la grosse boule noire présente au centre du niveau. C'est un ennemi, attaché par une chaîne à son foyer. Si Mario s'en approche trop, le Chomp se rue tout droit vers Mario, jusqu'à être bloqué par la longueur de sa chaîne.

Le Chomp est modélisé sous la forme d'une hiérarchie, contenant notamment deux demi-sphères pour que les deux machoires puissent s'ouvrir et se fermer. Aucun modèle n'a été utilisé pour la modélisation, mais les textures ont été importées. Comme il s'agissait de plusieurs images, cela nous a incités à ajouter une fonction `draw_hierarchy_element` aux côtés des autres fonctions `draw` de VCL et ainsi pouvoir appliquer une texture différente pour chaque partie du corps du chomp.

Nous avons programmé pour le Chomp un comportement fidèle au jeu. En mode passif, le chomp fait des petits mouvements aléatoires en restant dans un rayon autour de son foyer. Quand Mario s'approche, le Chomp se tourne rapidement vers Mario et continue de le guetter tant que Mario ne s'éloigne pas d'un certain rayon. Au bout de 2 secondes le Chomp entame une ruée vers Mario, et se fait arrêter à la limite de son rayon d'activité pour simuler des chaînes qui le maintiennent proche de son foyer. Il reste figé dans sa position limite 1 seconde puis retombe par simulation de gravité jusqu'à atteindre la hauteur du sol, constante autour du Chomp, donc il

n'y a pas besoin de calculer des collisions pour cela.

Les 4 chaînes visuelles du Chomp possèdent leur propre modèle physique. Chaque chaîne "traîne" derrière la chaîne un cran plus extérieure au foyer (la plus extérieure subissant les déplacements du Chomp) en veillant à garder une distance maximale entre deux éléments consécutifs d'un cinquième du rayon maximal Foyer-Chomp. Si un déplacement d'un élément A vient à la placer au delà de cette distance de la chaîne suivante B, plus intérieure, B se déplace vers A pour repasser sous la distance limite. Exception faite pour le foyer qui est fixe : si la chaîne la plus intérieure s'éloigne trop du foyer, c'est à toutes les chaînes qu'est appliqué un petit rapprochement vers le foyer. Comme ce dernier rapprochement vers le foyer est à contre-sens du rapprochement vers le Chomp, il faut simuler ces étapes quelques fois (moins de 10 fois) jusqu'à observer une convergence.

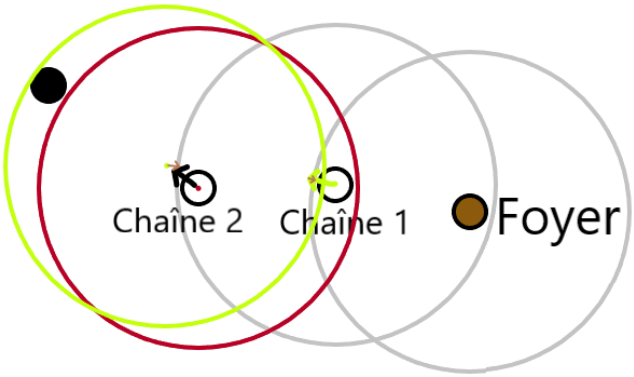


FIGURE 1 – Après déplacement du Chomp en dehors du rayon de la dernière chaîne, cette chaîne se rapproche du Chomp (flèche noire), puis la chaîne précédente se rapproche de la dernière chaîne (flèche verte) etc... puis, si besoin, le foyer force le rapprochement vers lui de toutes les chaînes (petites flèches marron)

2.2 Terrain

Pour reproduire le terrain du niveau *Bob-omb Battlefield*, nous avons importé un modèle, contenant un fichier .obj et un fichier .mtl pour faire le lien avec plusieurs textures. Vu le nombre d'images que cela représentait, nous avons choisi de coder notre propre parseur de OBJ/MTL pour finalement stocker un `mesh_drawable` par texture, afin de pouvoir afficher le niveau entier en une simple boucle for, tout en limitant au maximum les appels à `glBindTexture` qui semble relativement lent.

2.3 Mario

Pour la création du modèle, nous sommes partis d'un modèle importé sur lequel nous avons rajouté quelques rectangles afin d'y ajouter les bandes rouges, les boutons, la ceinture et les épaulettes du GU. Nous avons également rajouté un cylindre en guise de cou à Mario, afin d'ajouter le col du GU. Le bicorne provient d'une modélisation sous Blender faite par le JTX. Le nombre important de

triangle de ce modèle nous a poussé à procéder à une décimation des triangles (https ://github.com/sp4cerat/Fast-Quadric-Mesh-Simplification). L'ensemble du modèle se retrouve alors stocké dans un fichier OBJ et MTL. Nous avons alors créé une fonction permettant de parser ces deux fichiers et de recréer la hierarchie du modèle, chaque partie de la hiérarchie ayant un nom dans le fichier OBJ après le tag `g`.

Nous voulions avoir Mario dans la position de vol. Il s'agissait alors d'effectuer des rotations sur chacune des parties de la hiérarchie, ce qui a été fait simplement avec le logiciel 3D Builder.

FIGURE 2 – Mario en GU, en position de vol

Pour la trajectoire de vol de Mario, la position est interpolée par splines cubiques à partir d'une trentaine de positions de contrôle autour du niveau. La vitesse de Mario est rendue à peu près constante par la modification automatique des temps de contrôle en fonction des positions de contrôle, pour laisser proportionnellement plus de temps entre deux positions plus éloignées. En plus de la vitesse et donc de l'orientation de Mario, nous avons appliqué une rotation selon l'axe pied/tête à Mario pour simuler un roulis lors des virages en vol, grâce à la dérivée seconde de sa position interpolée. Les splines utilisées n'étant que de classe C^1 aux positions de contrôle, nous avons borné l'effet de la dérivée seconde et choisi avec attention les positions clés de Mario de sorte à ce que les discontinuités ne s'observent pas.

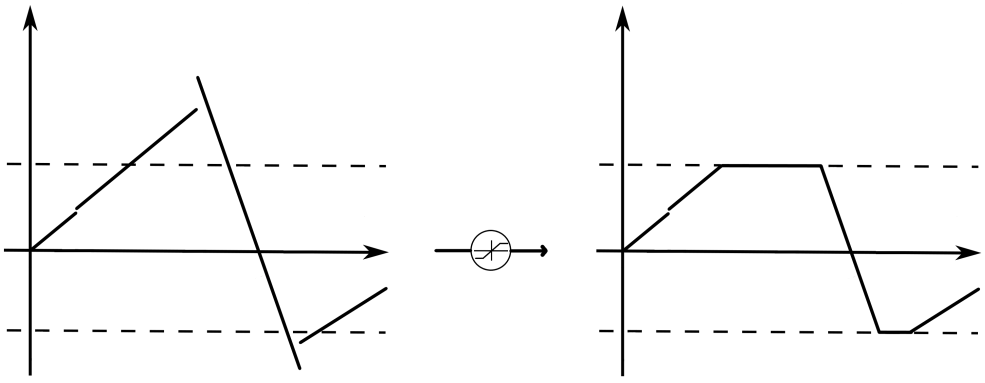


FIGURE 3 – Pour obtenir l'angle de roulis de Mario, appliquer un simple filtre permet effectivement de gommer la plupart des discontinuités de dérivée seconde de la spline

2.4 Collisions

Pour implémenter les collisions, nous quadrillons le terrain en 32*32 blocs. Tester la collision d'un cylindre ou d'une boule (toutes les hitboxes du jeu sont de ces deux formes) avec le terrain c'est tester la collision de ces figures avec l'ensemble des triangles qui sont dans le bloc contenant la figure (ou les blocs si la figure est à une frontière entre blocs). Avec ce quadrillage, le nombre de triangles à tester est très bas et les performances ne sont pas impactées.

Plus précisément, nous testons les collisions des bulles et des bobombs avec les triangles du terrain et du pont. Les hitboxes dépendent du type de triangle. Dans notre scène, comme dans le jeu, les triangles sont soit des sols, soit des murs, soit des plafonds selon la coordonnée verticale de leur normale.

Pour tester la collision d'un objet désigné par un point P avec un triangle de sol, on vérifie si un segment vertical contenant P (de longueur prédéterminée) intersecte le triangle, ce qui revient à vérifier si le point se trouve dans une région en forme de prisme qui s'étend un peu au dessus et en dessous du sol (cf schéma). Si le point est trouvé dans le prisme, on le plaque à la surface du triangle par une projection verticale (même si le sol n'est pas totalement horizontal). Cela permet de monter des pentes et même des petites marches d'escalier et de ne pas constamment sautiller en marchant sur une pente descendante.

Un triangle de mur repousse un objet qui lui rentre dedans, d'une distance donnée en argument des fonctions de collision de mur. Typiquement, avec le rayon d'un objet circulaire, le centre de l'objet se fait repousser de façon à faire croire que c'est le bord de l'objet qui touche le mur. Enfin, un triangle de plafond, dont la collision est seulement utilisée par les Bob-ombs, agit similairement aux murs et de plus annule la vitesse verticale d'un objet qui le touche, alors qu'un objet en phase montante qui rencontre un mur peut continuer à monter dans son élan. Ces effets de collisions sont relativement bien fidèles au jeu.

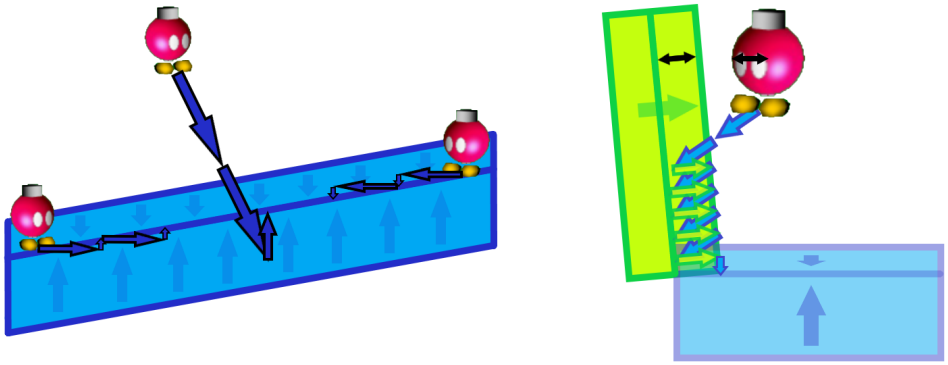


FIGURE 4 – Les collisions sur les murs et les sols se distinguent par le caractère "répulsif" du premier et "attractif" du second

2.5 Bulle

La bulle est l'objet sphérique qui tombe périodiquement dans la zone du niveau devant la grande grille. Dans le jeu, la bulle n'apparaît que si Mario se trouve dans cette zone du niveau. Une bulle apparaît à un point fixe, invisible car en hauteur, a une vitesse initiale qui vise Mario mais sinon chute naturellement (gravité et frottements). Au contact du sol, l'objet sphérique s'applatit et s'élargit, avant de repartir à la normale et rebondir, pour simuler un effet d'élan dans la chute. La bulle change de direction pour viser Mario à chaque rebond, et explose finalement au 3e impact avec le sol.

Nous avons pu reproduire ce comportement avec le Bob-omb rose à la place de Mario (dans notre scène, c'est ce Bob-omb rose qu'on contrôle au clavier). Nos structures de Bob-ombs et bulles étant distinctes, pour donner à la bulle l'accès à la position du Bob-omb rose, on donne simplement cette position en argument de la fonction calculant l'évolution physique de la bulle.

En plus de ce comportement, nous avons permis aux bulles de continuer d'apparaître périodiquement lorsque le Bob-omb rose se trouve loin de la zone où les bulles apparaissent dans le jeu. Dans ce cas, les bulles tombent sous les seules loi de la physique : gravité et frottements proportionnels à la vitesse pour la chute, direction après rebond respectant la normale du triangle d'impact au sol, calculée grâce aux fonctions de collision.

Visuellement, la bulle possède une texture la rendant bleu ciel en haut et vert herbe en bas, pour faire croire à un objet transparent réfléchissant. Pour notre scène, nous avons créé notre propre texture en ajoutant des faux nuages.

FIGURE 5 – De gauche à droite : la bulle du jeu ; notre texture ; rendu final

2.6 Pont

Le pont, au centre du niveau près du Chomp, modélisé comme un pavé, subit des forces extérieures qui peuvent le faire balancer d'un côté ou d'un autre, des forces internes qui le poussent à revenir à une position d'équilibre horizontale, et indépendamment de ces forces le pont produit toujours une petite oscillation, rajoutée artificiellement pour garder la scène vivante même sans forces extérieures.

Du côté interne, la rotation de notre pont est similaire à celle d'un ressort à torsion : le pont subit des forces qui le poussent à revenir à l'horizontale, qui est sa position d'équilibre, tout en subissant des forces qui résistent au mouvement, ce qui rend la position d'équilibre stable, avec convergence après un nombre d'oscillations qui dépend des paramètres physiques que l'on a choisis.

Si un Bob-omb marche sur le pont, une collision est enregistrée et met à jour la variable stockant les moments appliqués au pont, proportionnellement à la distance signée entre le centre du pont et le point d'impact, tel un vrai moment cinétique. Le code d'une itération de simulation physique du pont ressemble alors à cela :

```
// phi: angle due to physical simulation
// theta: small, artificial, periodic oscillations
// sigma: external momentum (bobombs walking on the bridge)
// tau: torsion momentum (resistance to oscillation)
sigma = 0.0;
for (collision (pt3D impact, mass m): collisions)
    sigma += cross(impact-center, rotation_axis).z * m*g;
tau = -kappa * phi - f * dphi;

// Angular momentum theorem: constant*d2phi/dt2 = sigma+tau
d2phi = (sigma + tau) / L;
dphi += d2phi * dt;
phi += dphi * dt;
theta = thetamax * sin(2 * PI * t / period);

set_bridge_angle(phi+theta);
```

2.7 Bobombs

Les bobombs (noirs et rose) sont composées d'une partie solide et de billboards. Les pieds et le boulons sont solides et le corps et les yeux sont des billboards. Les pieds sont construits à partir d'un ensemble de sommets qui initialement était décrit dans un fichier OBJ. Quant au boulon, il s'agit d'un prisme à base hexagonale.

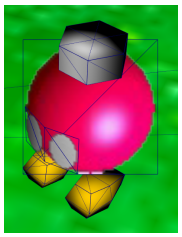
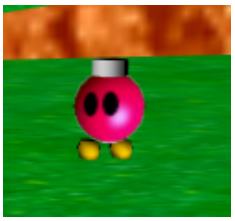


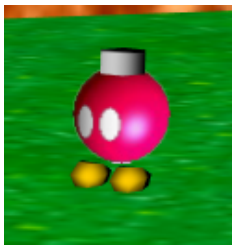
FIGURE 6 – Vue en wireframe d'un Bob-omb

Le corps doit avoir la même forme circulaire quel que soit l'angle de la caméra, nous l'affichons donc toujours face à celle-ci. Les yeux sont affichés sur deux rectangles qui sont placés à une distance d'un rayon du corps de la Bob-omb de son

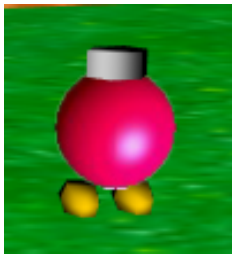
centre, pour renforcer l'impression que le corps n'est pas un billboard. Pour l'affichage de ces deux billboards, afin d'éviter des bugs d'affichage, nous prenons en compte la distance à laquelle se trouve le centre du corps et le centre des yeux de la caméra, et nous n'affichons les yeux que lorsqu'ils sont les plus proches.



(a) Sans correctif, les yeux sont derrière le corps



(b) Les yeux sont devant le corps, affichage normal

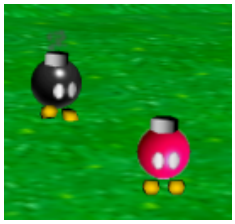


(c) Avec correctif, les yeux sont derrière le corps

FIGURE 7 – Bug d'affichage des billboards sur les Bob-ombs

Lorsque qu'un Bob-omb se déplace, les pieds subissent une rotation sinusoïdale, donnant l'impression que ceux-ci se lèvent et s'abaissent successivement. La fréquence et l'amplitude de ce mouvement dépendent de l'état dans lequel se trouve le Bob-omb. S'il est passif dans l'attente de l'approche du Bob-omb rose, l'amplitude et la fréquence sont faibles, au contraire lorsque le Bob-omb est en mode course, l'amplitude et la fréquence sont élevées pour accentuer l'impression de poursuite.

Un Bob-omb noir se déplace aléatoirement autour de sa position initiale en attendant que le Bob-omb rose s'en approche. Lorsque celle-ci est assez proche (quelques rayons d'un Bob-omb) le Bob-omb noir rentre dans un mode de course, se déplaçant à vitesse fixe, en suivant le Bob-omb rose, un billboard de fumée tournoyant s'échappant du boulon. Après quelques secondes de course, le Bob-omb disparait pour laisser 14 billboards d'une explosion s'afficher successivement sur une demie seconde. Le Bob-omb n'est alors plus affiché pendant une dizaine de secondes avant d'être replacé à sa position initiale dans l'état passif.



(a) Poursuite du Bob-omb rose par un Bob-omb noir, fumée

(b) Explosion d'un Bob-omb noir

2.8 Interactivité

Une fois les collisions implémentées, nous avons souhaité rendre le Bob-omb rose contrôlable au clavier, Mario étant déjà en circuit de vol. Au clavier AZERTY

(resp. QWERTY), les touches ZQSD (resp. WASD) permettent respectivement au Bob-omb rose d'avancer, tourner à gauche, reculer, tourner à droite. La touche espace permet de faire sauter le personnage. Pour assurer une meilleure immersion, nous avons ajouté au GUI l'option de pouvoir centrer la caméra sur le Bob-omb rose, et de la tourner derrière lui pour suivre ses mouvements. Pour ce faire, nous avons utilisé les fonctions disponibles `scene.camera.apply_translation` et `apply_rotation` sur les position et orientation du Bob-omb rose.

2.9 Divers

Les arbres et les pièces du niveau sont des billboards qui modifient leur orientation en fonction de la caméra pour toujours apparaître de face, comme les corps des Bob-ombs. Les pièces sont en réalité constituées de 4 billboards qui se succèdent de manière cyclique pour donner une impression de rotation, partagés par les deux types de pièces, qui sont différenciés par leur paramètre uniforme de couleur.

3 Conclusion

Ce projet nous aura donné un aperçu des nombreux défis qui se posent aux créateurs de jeux vidéo. Notamment, de la difficulté de passer de la théorie de l'intersection d'une droite et un triangle, à la pratique de la gestion effective des collisions avec plusieurs triangles en même temps, qu'ils soient deux murs, ou un sol et un plafond un peu trop proches.

Egalement, le défi de la performance. Notre scène comporte de très nombreux billboards de types différents, et nous avons fait le choix de ne centraliser que partiellement l'affichage des billboards pour limiter les appels aux fonctions critiques, au détriment de la cohérence des distances lors de l'affichage final.

Du côté de la modélisation, même les minorités des modèles et de textures de notre scène que nous n'avons pas importés nous ont pris beaucoup de temps à peaufiner.

Enfin, les comportements et IA, en demandant des codes à rallonge pleins de if et else entremêlés se sont révélés être des parties à fort potentiel de bug.

Ce projet nous a permis de découvrir des astuces employées pour répondre à tous ces défis, une recherche stimulante qui se solde par une scène finale dont notre satisfaction n'égale que la difficulté à tout synthétiser dans ce rapport.