



Promotion X2019 - Année académique 2020/2021

---

Josselin SOMERVILLE ROBERTS  
Charbel HOBEIKA  
Zahi EL HAJJ



## 1 Introduction et description du concept

Ce projet consiste à contrôler un loup en troisième personne qui peut se déplacer dans toutes les directions. L'utilisateur peut se promener dans différentes zones géographiques : un désert, une forêt magique, une forêt, une prairie et un village.

Le champ de vision du loup diminue au cours du temps, ce qui impose à l'utilisateur de trouver le bon chemin jusqu'aux *runes*. Une fois à proximité d'une *rune*, l'utilisateur peut appuyer sur le **bouton A** pour recharger son champ de vision. L'utilisateur a cependant la possibilité d'appuyer sur le **bouton Q** pour augmenter massivement la lumière autour du loup, diminuant en contrepartie plus vite le rayon de son cercle de vision.

L'utilisateur peut ainsi découvrir le monde, en profitant de la grandeur des pyramides dans le désert, puis en se faufilant dans une forêt magique au sein de lumières envoûtantes. Après ces différents voyages, l'utilisateur peut (et devrait) aller se rafraîchir au sein d'une forêt sereine à proximité d'un village où il trouvera des tours et des maisons, des ruines pour recharger sa vision et un feu de camp pour bien se réchauffer et se reposer.



Figure 1: Loup qui recharge sa vision

Le concept de champ de vision limité a été inspiré du projet *Farewell North*.

## 2 Génération du terrain

Le terrain est généré de façon entièrement procédurale et découpé en *chunks* ce qui permet d'afficher seulement la partie que l'on voit. Le relief est généré grâce à un bruit de Perlin, ce qui a pour avantage d'ignorer les problèmes de continuité entre chunks qui pourraient être visibles. Un autre avantage est que quand une partie du terrain disparaît de la vue du personnage, on n'a pas à stocker les points du maillage en mémoire : les calculs effectués par la fonction du bruit de Perlin donnent toujours les mêmes résultats. Ceci permet donc d'éviter les incohérences spatiales suite au déplacement du personnage.

Ensuite, pour peupler le terrain, nous avons eu recours à deux bruits de Perlin d'échelle différente:

- un à basse fréquence : pour représenter le type de la zone, qui varie entre forêt, prairie, village, désert ou encore forêt magique.
- un à haute fréquence : pour générer les éléments individuels : dans le cas de la forêt les pierres, les arbres... et dans le cas du village les maisons, les lampadaires...

La texture du sol est définie selon la zone. En plus des éléments *simples*, on retrouve des structures particulières telles que des pyramides ou des tours. Des oiseaux volent autour de ces grandes structures et même parfois dans la forêt.

### 3 Génération des éléments simples

Plusieurs types d'arbres ont été utilisés et ont été téléchargés aussi grâce au site *sketchfab* et traités sur *Blender*. On distingue deux types de bosquets pour la forêt normale :

- un bosquet de chênes
- un bosquet de sapins

Un bruit de Perlin grossier a été utilisé pour déterminer le type de bosquet, pour ne pas se retrouver avec des chênes et des sapins ensemble. Un bruit de Perlin plus fin a ensuite été utilisé pour modéliser les différents types de sapins ou de chênes au sein d'un même bosquet. On retrouve le même principe pour les cactus dans le désert, les champignons dans la forêt magique et les maisons dans les villages.

Pour calculer l'emplacement des arbres, on a utilisé l'algorithme *d'échantillonnage par disque de poisson*, qui permet de générer des positions aléatoires, espacées d'une distance minimale, tout en ayant une complexité linéaire, ce qui est bien moins coûteux qu'un simple test avec toutes les positions déjà occupées, notamment sur un terrain très dense.

Pour l'orientation des différents arbres, et pour obtenir un rendu plus réaliste et pour éviter les alignements qui sautent à l'oeil, une moyenne entre la normale au terrain et la verticale a été choisie comme direction de chaque arbre. Si la composante selon l'**axe z** est trop faible (pente trop élevée), les arbres ne sont cependant pas plantés.

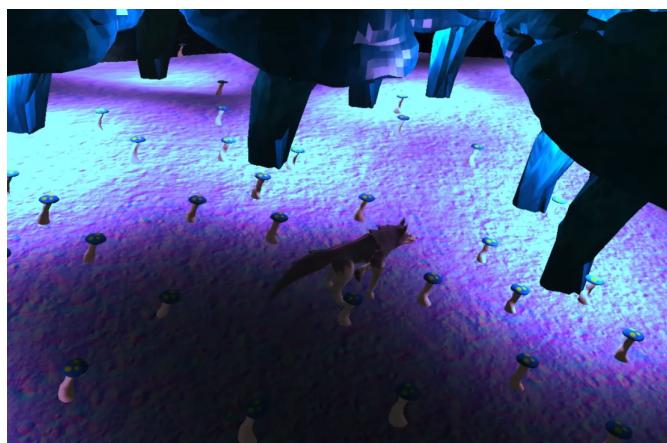


Figure 2: Forêt magique

## 4 Génération des grandes structures

L'utilisateur peut trouver dans certaines parties du monde des structures larges telles que les pyramides. Ces pyramides sont disposées aléatoirement sur le terrain. Une difficulté de ces structures est leur taille qui est supérieure à celle d'un *chunk*. Afin d'éviter les intersections d'objets tels que des cactus avec ces structures, nous avons choisi de les générer dans des endroits *peu peuplés*. Pour ce faire, une grande plage de valeur dans le bruit de Perlin est réservé à une structure telle qu'une pyramide, ce qui assure qu'aucun cactus ne sera généré à proximité.



Figure 3: Pyramide

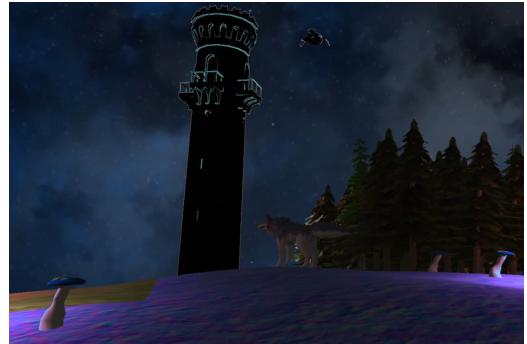


Figure 4: Tour

## 5 Utilisation des *shaders*

Notre rendu a été amélioré par l'utilisation de *shaders* permettant de nombreuses améliorations graphiques. Nous souhaitions avoir un rendu prenant en compte la transparence, les sources de lumière, le brouillard, un fond de type *skybox* et enfin bien sur la vision du loup très particulière. Afin de bien différencier la zone que le coup voit nettement et celle qui ne voit pas bien, nous avons décidé d'afficher seulement les contours des objets au loin et de façon très sombre. La méthode utilisée est celle de filtrage par convolution, qui permet de détecter le contour d'une image. Le rendu est alors effectué en plusieurs étapes : un premier passage permet de créer la texture des objets lointains qui sont filtrés. Ensuite, un deuxième passage permet de créer une texture correspondant aux objets proches. Enfin, ces deux textures sont assignées à des quadrangles afin de les superposer et d'afficher la scène.

Les *shaders* prennent aussi en compte la distance du loup par rapport aux autres objets pour pouvoir traiter les contours des objets à proximité. Pour résoudre le problème d'objets comme l'herbe, une texture plus opaque a été utilisée pour éviter les défauts de rendu de son contour.

Les sources de lumières ont été définies de sorte qu'on puisse calculer l'illumination en fonction de leur distance par rapport au loup.

Des effets de transparence, notamment pour les feuilles d'arbres, et de brouillard ont aussi été utilisés.

## 6 Animation du Loup

Pour la modélisation du loup, nous avons téléchargé un modèle 3D sur le site *sketchfab*. Ensuite, on a découpé le modèle en plusieurs parties sur le logiciel *Blender*. Ce découpage, a été fait à la main

pour être le plus précis possible afin d'éviter des problèmes visuels. Ceci nous a permis d'animer le loup selon la structure d'un *squelette articulé*.

L'utilisateur contrôle le loup et peut le diriger à l'aide des flèches du clavier. L'utilisateur peut même appuyer sur le bouton *Space* et le loup commencera à courir. Suite à ce changement de vitesse de course :

- les positions d'interpolation pour l'animation changent, avec des plis de pattes plus accentués en course. Cependant, cette amplification du mouvement s'accompagne aussi d'une amplification des problèmes de raccordement au niveau des articulations dus au découpage à la main des parties du loup.
- la synchronisation du mouvement des pattes est différente : on passe d'un quart de période entre chaque patte, à une demi période entre les deux pattes avant et les deux pattes arrière. Ceci permet à l'utilisateur de mieux cerner la différence du mouvement au delà du fait que le loup change de vitesse.
- les mouvements de la queue et du cou sont aussi adaptés à la vitesse du loup.

Afin de positionner le loup sur le terrain, nous avons implémenté une méthode qui consiste à interpoler les points du *chunk* sur lequel le loup se situe.

Pour l'orientation du loup, on utilise la composante selon **l'axe z** de la normale au terrain, en imposant une contrainte de blocage du mouvement une fois que cette composante z est trop faible (pente trop forte).

Cependant, on ne prend pas en compte la collision du loup avec les autres objets de la scène.

## 7 Animation des Sprites

Pour l'animation du feu de camp, nous avons utilisé une méthode basée sur des sprites animés plaqués sur un *billboard*. Cette texture 2D est une image qui regroupe des formes qui défilent au cours du temps. Pour donner un effet volumique au feu, plusieurs quadrangles ont été affichés en effectuant une rotation à pas constant.

La même méthode a été utilisée pour représenter et animer l'effet autour des différentes *runes* qui servent d'un puits pour recharger sa vision. Le quadrangle utilisé est translaté au cours du temps, pour envelopper la *rune* sur toute sa longueur.



Figure 5: Sprite du feu

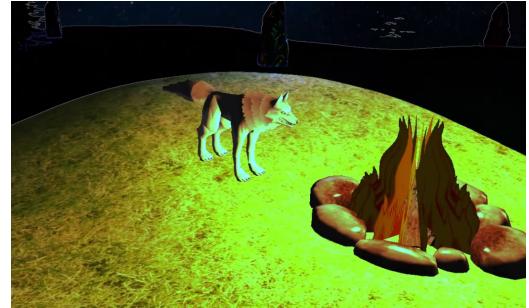


Figure 6: Rendu des billboards

## 8 Caméra

L'utilisateur possède une vue de troisième personne, et a la possibilité de tourner autour du loup ainsi que de *zoomer* et de *dezoomer*. Pour que l'expérience de l'utilisateur soit plus fluide, les déplacements sont filtrés :

- le mouvement de la caméra, qui suit toujours le loup, est atténué de sorte que ça ne soit pas trop brusque en parcourant les bosses du terrain, surtout quand la vitesse du loup augmente.
- le mouvement de la caméra est aussi fluidifié si l'utilisateur la déplace très brusquement.

## 9 Conclusion

En conclusion, le joueur contrôle un loup, et peut se balader afin de découvrir différentes structures. Ce projet nous a permis de mettre en relation les connaissances apprises en cours avec les difficultés d'optimisation auxquelles nous avons fait face. Il a donc fallu faire preuve de créativité et d'ingéniosité pour développer des méthodes efficaces telles que la segmentation en *chunks*, l'utilisation de l'échantillonnage par disque de poisson ou encore le chargement optimisé des textures.