Home

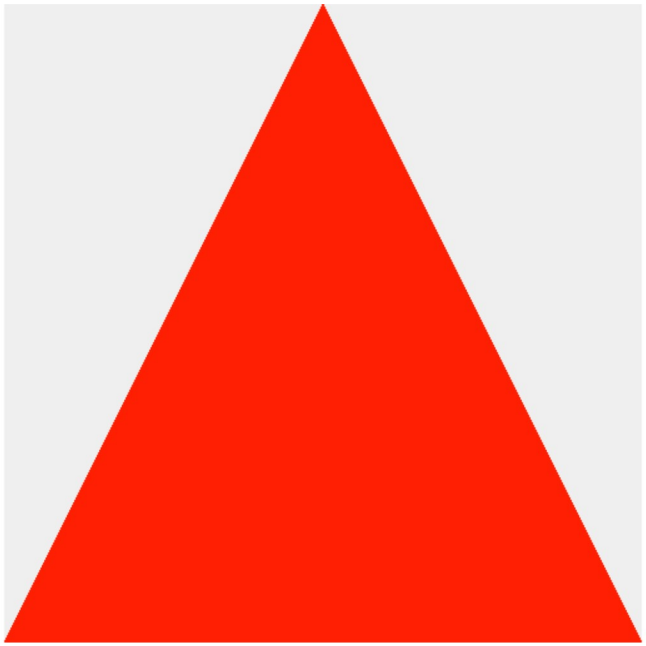
Un ciel dans le shader.

28/06/2015 - Simon Rodriguez

J'ai eu l'occasion durant ce semestre de commencer à apprendre à programmer pour OpenGL. Il s'agit d'une interface de programmation graphique implémentée par la plupart des fabricants de cartes graphiques, et qui permet d'effectuer des rendus 2D/3D en temps réel [1]. OpenGL est composée de deux parties :

- un ensemble de fonctions de référence qui permettent de communiquer avec le processeur graphique au sein d'un programme, pour lui transférer des objets à afficher [2], des textures, etc.
- et le langage GLSL destiné aux *shaders*, petits programmes qui sont executés directement sur la carte graphique, et pouvant être appelés pour chaque sommet d'un objet (*vertex shader*), ou pour chaque pixel de l'image à rendre (*fragment shader*).

OpenGL est relativement bas-niveau, ce qui permet faire beaucoup de choses et d'avoir le contrôle sur chaque étape du rendu. On peut ainsi faire ceci :



ou cela :

Contrairement à un moteur de rendu comme Unity ou Unreal Engine, rien n'est implémenté au-delà du pipeline de rendu : à charge pour le développeur de générer ou charger les objets, les transférer à la partie graphique, lui indiquer comment les rendre, avec quelles textures, quels réglages, etc. [3] On a ainsi plus de contrôle sur le processus de rendu et les shaders.

Pour le projet final de ce cours, qui consistait à réaliser un programme permettant d'explorer un terrain généré aléatoirement, avec textures, éclairage, animations, reflets et vagues réalistes, mouvements de caméra, etc., j'ai développé un modèle de ciel procédural simple à mettre en place

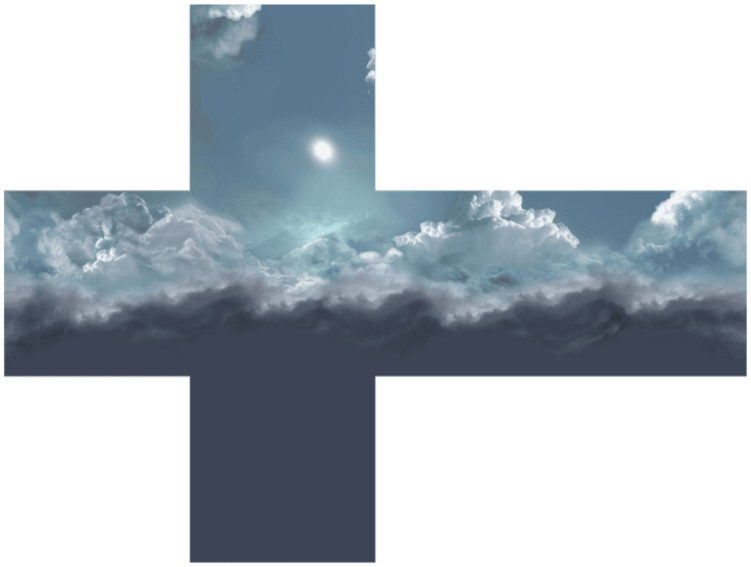
et assez efficace.

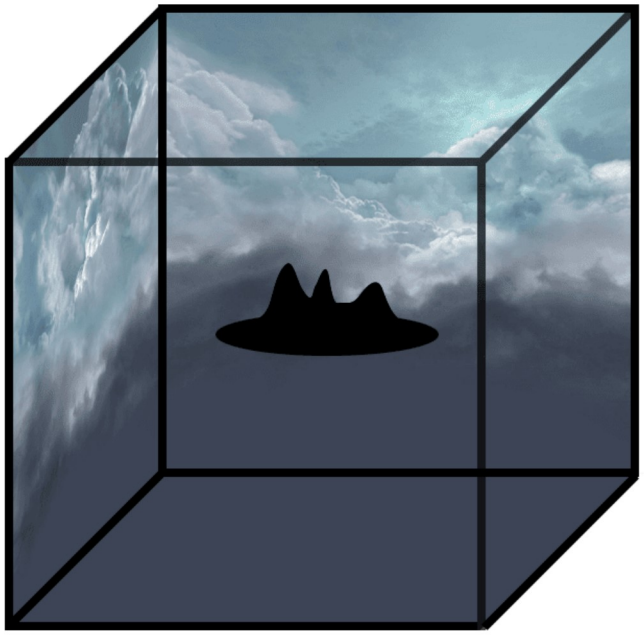
Mon idée générale était de créer un ciel avec certains pré-requis :

- procédural (pas une simple image chargée depuis le disque)
- capable de changer selon l'heure
- comprenant le soleil, la lune, les étoiles
- animé
- avec des nuages
- et une météo modifiable
- rapide et pas trop coûteux

Skythings

Tout d'abord, une bref présentation autour du thème "mais comment faire un ciel ?". La solution la plus simple est ce qu'on appelle une *skybox*. Il suffit de placer l'ensemble de sa scène à l'intérieur d'un immense cube, et de texturer ce dernier avec un panorama. Simple à mettre en oeuvre et rapide. Le seul souci est que la forme de cube s'adapte mal pour réaliser un ciel procédural, où l'on souhaite pouvoir mixer des couleurs en se basant sur la hauteur de chaque point du ciel, hauteur qui ne varie pas de façon très régulière le long des arêtes d'un cube.





Une autre possibilité est d'utiliser un *skydome*, autrement dit remplacer le cube par une sphère avec un nombre de sommets suffisant pour avoir quelque chose de propre. La sphère se prête très bien à des constructions procédurales, moins bien à des textures toutes faites [4].

En choisissant de se concentrer sur un ciel procédural, deux possibilités s'offraient à moi :

- faire du procédural jusqu'au bout en utilisant un modèle physique de dispersion de la lumière dans l'atmosphère, tel la diffusion de Mie ou de Raylegh [5]. Mais ces modèles sont complexes et coûteux, avec beaucoup de calculs à effectuer dans le fragment shader associé au skydome.
- utiliser des tables de couleurs pour créer des teintes dépendant de l'heure de la journée, de la météo, de la hauteur de chaque point du ciel. Beaucoup moins de calculs, quelques textures en plus à créer. Mes deux principales inspirations sur le sujet sont une implémentation dans un projet similaire, et un article sur l'utilisation de telles techniques (combinées avec d'autres notions d'éclairage plus complexes) dans Far Cry

3.

Implémentation

La majeure partie de l'implémentation se déroule dans le fragment shader. La partie C++ du code sert juste à attacher les textures de références et à mettre à jour la position du soleil, les matrices de transformations de la scène et certaines valeurs numériques.
En entrée du shader, nous avons :

- la position normalisée du soleil, `sun_norm`
- la position normalisée du fragment courant, `pos_norm`
- le temps en secondes, `time`
- `weather`, un nombre entre 0.5 et 1.0 qui détermine s'il fait beau (`weather=1`) ou pas
- des textures pour le ciel (`tint` et `tint2`), pour le soleil (`sun`), la lune (`moon`) et les nuages (`clouds1` et `clouds2`)

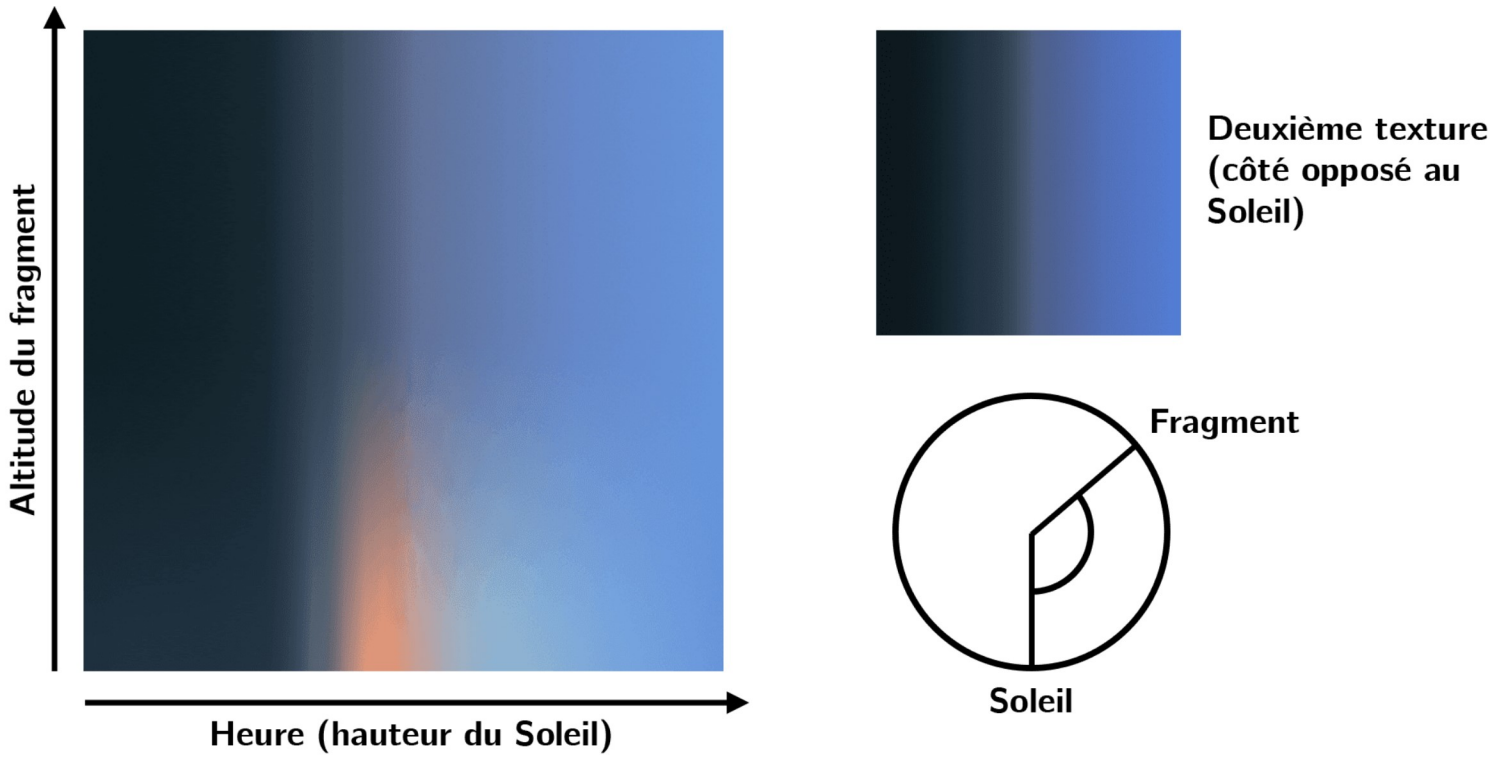
De plus, on suppose que la rotation des étoiles a déjà été effectuée dans le vertex shader en utilisant une matrice de rotation uniforme, et on a donc en entrée du fragment shader `star_pos` qui correspond a la projection du fragment dans l'espace de coordonnées des étoiles (qui représente une rotation de la sphère céleste autour d'un axe nord-sud incliné à 23° environ).

Toutes les coordonnées sont dans l'espace de la scène (*world coordinates*), ie avant passage dans l'espace caméra et la projection.

La sortie est `color`, un vecteur contenant les composantes rouge, vert, bleu et alpha (transparence) du fragment considéré.

Couleur du ciel

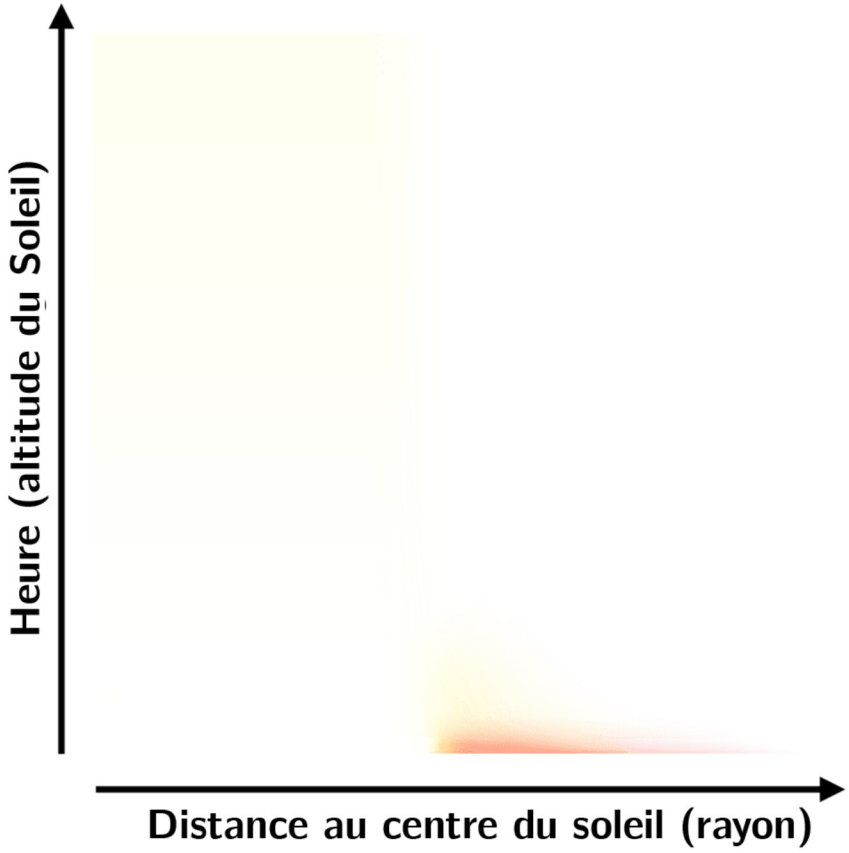
Pour ajuster la couleur du ciel en fonction de l'heure de la journée, on peut se baser sur la hauteur du soleil au-dessus de l'horizon. La coloration de chaque fragment dépend de plus de son altitude (à cause de la dispersion et de l'épaisseur de l'atmosphère lorsque l'on regarde vers l'horizon).
J'utilise ainsi une texture dans laquelle le shader lit, où l'abscisse correspond à la hauteur du soleil dans le ciel et l'ordonnée à l'altitude du fragment considéré.

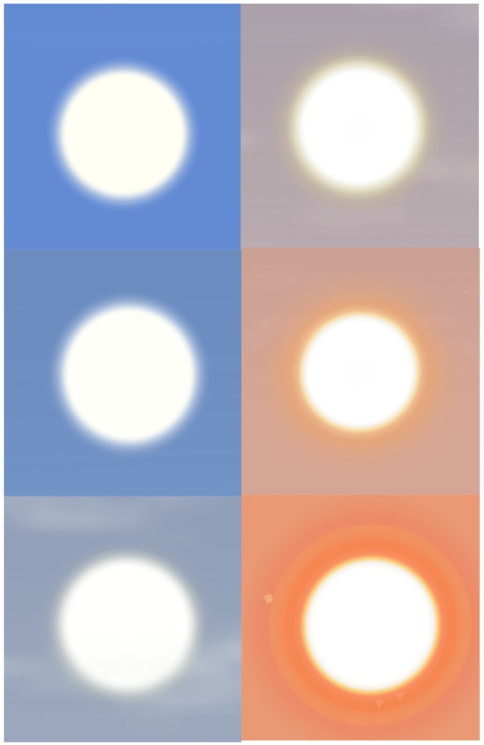


Pour plus de réalisme, on introduit deux textures de ce type : une utilisée pour colorer la demi-sphère centrée sur le Soleil, contenant des effets rouge-orangé pour les lever et coucher de Soleil, et une texture plus neutre pour la demi-sphère opposée. Les deux textures sont mélangées en se basant sur l'angle formé par le Soleil, le centre de la sphère, et la position du fragment, dans le plan horizontal.

Le soleil

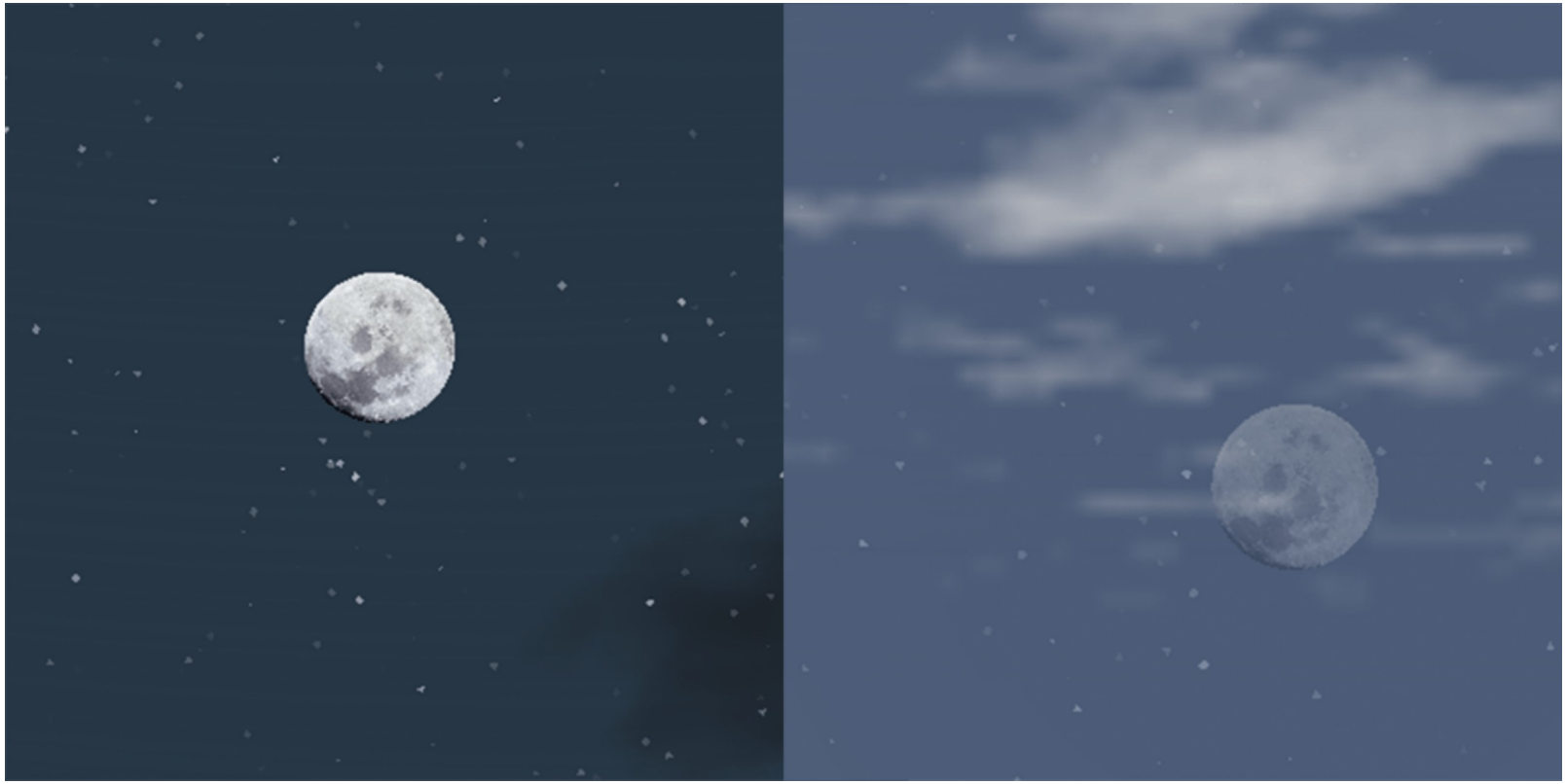
Le Soleil est généré en déterminant les fragments suffisamment proches de la position du Soleil pour faire partie de l'astre. Pour ces fragments, la couleur est prélevée dans une texture où l'abscisse correspond au rayon (distance du fragment au centre du soleil), et l'ordonnée à l'heure de la journée. On peut ainsi varier la couleur, le rayon et le halo entourant le Soleil. Cependant c'est à mon avis la partie la moins satisfaisante du shader, les couleurs et les effets de halos étant difficiles à reproduire en leur donnant une apparence naturelle.





La lune

Pour la Lune, l'idée est la même que pour le Soleil : déterminer si le fragment courant est suffisamement proche du centre de la Lune (qui est à l'opposé du centre du Soleil), et si oui, lire dans une texture la couleur à utiliser. Cependant, la texture de la Lune est une image "complète", et pas juste une couleur en fonction du rayon et de la position du soleil. Il y a donc une projection à effectuer pour savoir où lire exactement dans la texture et ne pas déformer l'image de la surface lunaire. Lors des levers et coucher de soleil, la Lune est rendue légèrement transparente pour tenir compte de la diffusion.



Les étoiles

Les étoiles sont générées aléatoirement, en utilisant les coordonnées de chaque point de la sphère comme hash pour générer le bruit. Seule les zones de très forte intensité sont conservées et colorées en blanc. En controlant ainsi le bruit pseudo-aléatoire, les étoiles sont toujours placées au même endroit sur la voûte celeste : une rotation en fonction du temps est préalablement appliquée pour animer le mouvement général des étoiles. Enfin, le mélange avec la couleur du ciel est ajustée en fonction de la position du Soleil.

Nuages et météo

Ma première tentative pour les nuages a été de générer du bruit basé sur le mouvement brownien fractal (*fBm noise*), en utilisant du bruit de Perlin comme composante de base. Mais le résultat était uniformément réparti en trois dimensions, donnant l'impression d'être au milieu d'une sphère de nuages, alors que dans le monde réél les nuages ont tendances à s'aligner sur des plans horizontaux. Cependant, projeter le bruit sur ce type de plans n'était pas plus convaincant, et j'ai finalement opté pour une solution à bases de textures.

J'ai ainsi généré dans Vue des panoramas de nuages blancs sur fond noir, avant de les projeter en coordonnées sphériques sur le dome. En décalant la coordonnée horizontale avec le temps, les nuages bougent horizontalement autour de la scène. Les valeurs (niveaux de gris) lues dans la texture servent à déterminer la transparence de la couche de nuage en tout point de la sphère. La couleur est ensuite calculée séparément et mélangée avec la couleur de ce qui se trouve derrière.

La météo est determinée par une valeur décimale `weather` qui oscille entre 1 (beau temps) et 0.5 (mauvais temps) [6]. Cette valeur est utilisé pour assombrir la couleur générale du ciel, déterminer la couleur des nuages et pour mixer les textures correspondant aux légers nuages de beau temps et aux gros nuages d'orage.

Conclusion

Ce shader pourrait bien entendu être amélioré, en utilisant un plus grand nombre de maps pour la teinte du ciel et pour la couleur et forme du soleil, et en ajoutant par exemple les phases de la Lune, des transitions plus crédibles entre les différents types de nuages, etc. Le code des shaders, le modèle de sphère et les ressources graphiques sont disponibles sur GitHub. Et pour ceux qui souhaiteraient voir le résultat du projet en entier, c'est ici.

1. tout comme DirectX sur Windows ↩
2. sous forme de listes de points, de propriétés et d'indices pour les relier de différentes façons ↩
3. heureusement des bibliothèques comme GLEW et GLUT sont là pour simplifier le travail. ↩
4. le mapping (application de la texture sur la géométrie) d'une sphère a toujours de petits soucis aux pôles ou ailleurs ↩
5. plus de détails ici ↩
6. pourquoi 0.5 et pas 0.0 ? parce que cette valeur est utilisée ailleurs dans le code (pour générer de la pluie notamment), et que je préfère garder une unique valeur cohérente ↩