1. On vous a fourni la fonction lineariserProfondeur(). Expliquez ce qu'accompli cette fonction.

La profondeur récupérée de la texture est en clip-space.

Dans cette situation la valeur de la profondeur suit la fonction présentée dans l'équation trouvée sur cette page: https://learnopengl.com/Advanced-OpenGL/Depth-testing

$$F_{depth} = \frac{\frac{1}{z} - \frac{1}{near}}{\frac{1}{far} - \frac{1}{near}}$$

L'on peut remarquer que cette équation n'est pas linéaire ce qui fait que les comparaisons seront plus ou moins précises selon la position dans la courbe en raison de la non-linéarité de la relation. La précision est moindre pour les distances élevées et la précision est élevée pour les courtes distances. Nous voulons éviter les transitions abruptes du floutage qui sont générées par l'imprécision des valeurs élevées.

L'on cherche à revenir à une relation linéaire telle que dans le view-space, c'est-à-dire, avant l'application de la matrice de projection. Le but est de bénéficier d'une précision adéquate sur l'ensemble de valeur possible et éviter les transitions abruptes de floutage.

2. On vous a aussi fourni la fonction FiltreGaussien(). Expliquez les détails de son fonctionnement.

Le filtre gaussien effectue une convolution sur le fragment avec un masque 5x5 d'écart-type 2.5 fourni. Étant donné un fragment, la fonction retourne la valeur convoluée de sa couleur originale.

Plus précisément, elle multiplie la valeur originale (échantillonnage dans le colormap) des 5x5 pixels autour du fragment par leur valeur respective dans le masque et fait la somme de ces résultats. L'étendue sert à échantillonner des valeurs plus loin. Si l'étendue est plus grande, cela a pour effet de flouter encore plus le résultat final, car plus les valeurs échantillonnées sont loin du fragment évalué, moins elles "ressemblent" à leur valeur originale.

3. Cette implémentation naïve du champ de profondeur effectue combien d'échantillonnage (sampling) de textures par fragment du quad plein écran ? Donner les détails sur une autre technique utilisée réduisant le nombre d'échantillonnage pour une même portée.

La fonction applique un masque 5x5. Pour un masque 5x5, il y a 25 échantillons à faire pour chaque pixel de l'écran. De plus, puisque la fonction est appelée deux fois, il y a 50 échantillons à faire pour chaque fragment de l'écran.

Une technique alternative est d'utilisé les compute shaders ainsi qu'une mémoire partagée. Le principe est que les résultats peuvent être réutilisés entre les fragments. Les compute shaders applique ainsi le filtre de gauss pour l'ensemble possible et sauvegarde le résultat dans la mémoire partagée. Le fragment shader lit ainsi à partir de la mémoire partagée les valeurs précalculées par les threads et utilise ces valeurs.

4. Avec notre implémentation présente du IBL, le gazon n'est pas visible dans les réflexions de l'environnement. Comment pourriez-vous régler ce problème en utilisant des FBOs ? Considérez ici un seul modèle 3D présent sur le gazon.

Le principe de la solution s'apparente à celui utilisé pour l'ombre de la statue sur le gazon. L'on veut réaliser une texture de profondeur et deux textures de couleurs selon une vue prise du modèle. Il y a une texture de couleur pour la composante spéculaire et une pour la composante diffuse.

- Il faut calculer la matrice MVP selon le centre du modèle.

- Réaliser une capture dans un FBO en dessinant le gazon selon cette matrice MVP.

- Le résultat de cette capture est les deux textures de couleurs et celle de profondeur selon le point de vue du modèle.

- Lors de l'appel à la fonction lightingIBL de modele3DFragment, il faut vérifier que la profondeur de la texture de profondeur générée pour le gazon est moindre que la valeur maximale, si c'est le cas l'on prend la couleur réfléchit à partir des textures de couleurs générées pour le gazon, sinon l'on prend la couleur réfléchit à partir des textures de couleurs pour la skybox.