Shadow Mapping

Description du problème

Lors du rendu d'une scène 3D de manière réaliste, la gestion de la lumière et des ombres est un aspect essentiel. En effet, sans ombres, une scène paraît plate, sans profondeur, et peu crédible.

Le shadow mapping est une technique couramment utilisée pour générer des ombres. Elle est appréciée pour sa flexibilité et sa compatibilité avec différents types de sources lumineuses. Le principe est relativement simple : il s'agit de « voir » la scène depuis le point de vue de la lumière afin de déterminer quelles zones sont visibles (et donc éclairées) et lesquelles sont cachées (et donc dans l'ombre).

Dans le cadre de ce projet, nous nous intéressons à l'implémentation du shadow mapping sur une scène statique. Nous chercherons à comprendre pourquoi cette méthode est adaptée à la génération d'ombres, comment elle fonctionne en pratique, quelles en sont les limites, et quelles améliorations peuvent être envisagées.

Manière de mesurer/visualiser le problème

Pour évaluer la qualité du shadow mapping dans une scène 3D statique, nous utilisons plusieurs techniques :

1. **Visualisation directe de la shadow map**
   Une première étape essentielle consiste à afficher la shadow map elle-même, c'est-à-dire l'image en niveaux de gris représentant la profondeur des objets vus depuis la lumière. Cela permet de vérifier que :

   - La scène est bien projetée dans l'espace lumière.
   - Les objets visibles sont bien enregistrés dans la profondeur.
   - Aucun objet n'est manquant ou mal positionné dans cette projection.

   Une shadow map correcte doit présenter des silhouettes cohérentes avec la géométrie réelle de la scène.

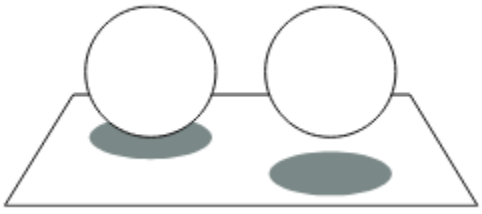
2. **Analyse visuelle du rendu final**
   Ensuite, on peut observer le rendu de la scène avec les ombres activées. On vérifie alors plusieurs points :
   - Les objets projettent bien des ombres au sol ou sur d'autres objets.
   - Les zones cachées par un objet par rapport à la lumière sont correctement assombries.
   - Il n'y a pas de shadow acne (des ombres parasites sur des surfaces éclairées).
   - Il n'y a pas d'ombres décalées de l'objet.

3. **Tests de position**
   Pour confirmer le bon fonctionnement de l'algorithme, on peut modifier la position :
   - **De la lumière** : une ombre cohérente doit changer dynamiquement selon la direction et la hauteur de la source lumineuse.
   - **Des objets** : déplacer un objet dans la scène doit modifier l'ombre projetée de façon réaliste.

Contexte

Le rendu réaliste des ombres est un élément crucial dans de nombreuses applications graphiques, notamment celles qui nécessitent une immersion visuelle forte ou une représentation précise de la scène. Le shadow mapping, bien que souvent utilisé dans des contextes temps réel, reste pertinent même dans le cadre de rendus statiques, car il illustre des principes fondamentaux du rendu d'ombres en 3D.

Regardons cette scène par exemple :



À première vue, l'objet à droite paraît plus proche, même si la sphère de gauche est en réalité plus grande (car elle est plus proche). Cela montre que les ombres jouent un rôle crucial dans notre perception des espaces 3D.

Voici quelques domaines où l'utilisation d'ombres réalistes est essentielle :

1. **Visualisation architecturale**
   Les architectes utilisent le rendu 3D pour simuler l'éclairage naturel et artificiel dans leurs projets. Les ombres permettent d'évaluer l'impact de la lumière à différents

moments de la journée, d'estimer les zones d'ombre dans un bâtiment, ou de tester l'orientation d'une structure.

2. **Prévisualisation de scènes 3D**
   Avant de lancer un rendu de haute qualité (par exemple en ray tracing), nous pouvons utiliser le shadow mapping pour avoir un aperçu rapide de la distribution des ombres dans une scène statique. Cela permet d'ajuster les sources de lumière ou la position des objets avant un rendu plus long et coûteux.

3. **Présentations techniques ou scientifiques**
   Dans les domaines comme la mécanique, l'aéronautique ou l'urbanisme, des rendus statiques sont souvent utilisés pour expliquer des concepts ou présenter des prototypes.

Explication de la méthode

Pour mettre en place le shadow mapping, nous avons commencé par simuler une source de lumière en suivant la même logique que celle utilisée pour la caméra dans le code source initial. La lumière possède donc, tout comme la caméra, une matrice de vue (viewMatrix), une matrice de projection (projMatrix) et une position dans l'espace. Cela nous permet de « voir » la scène du point de vue de la lumière.

Pour pouvoir effectuer le test d'ombre dans le fragment shader, chaque fragment a besoin de connaître sa position dans l'espace de la lumière. Cette information est calculée dès le vertex shader. Nous stockons la position du sommet dans l'espace clip de la lumière dans les indices 14 à 17 du vecteur de sortie.

Nous avons implémenté une fonction *generateShadowMap* qui se charge de générer la carte d'ombres. Cette fonction réalise plusieurs opérations :

1. **Transformation des sommets dans l'espace de la lumière**
   Chaque sommet de la scène est transformé à l'aide des matrices de la lumière (vue et projection), de la même manière que les sommets sont transformés pour le rendu depuis la caméra. On obtient ainsi les coordonnées clip space de chaque sommet vues depuis la lumière.

2. **Projection perspective et normalisation**
   Après la transformation, on applique une division par la composante *w* pour passer dans l'espace normalisé, ce qui permet de ramener les coordonnées dans un intervalle $[-1,1]$.

3. **Rasterization des triangles**
   À partir des sommets transformés, les triangles de la scène sont rasterisés à l'aide de la

fonction *Rasterizer* fournie dans le code initial. Cela permet d'obtenir les fragments visibles depuis la lumière, c'est-à-dire ceux qui ne sont pas cachés par d'autres objets.

4. **Remplissage de la shadow map**
   Pour chaque fragment généré, on stocke sa profondeur (la coordonnée *z*) dans une texture 2D représentant la shadow map. Si plusieurs fragments se projettent sur le même pixel, seule la profondeur la plus proche de la lumière est conservée.

Une fois la shadow map générée, nous l'utilisons dans le *fragmentShader* pour déterminer si un fragment se trouve dans l'ombre ou non.

L'idée est la suivante :

- Si, du point de vue de la lumière, un fragment est plus éloigné que ce qui a été enregistré dans la shadow map, cela signifie qu'il est masqué par un autre objet, donc il est dans l'ombre.

- Sinon, le fragment est visible depuis la lumière, il est éclairé.

Dans le shader, nous avons d'abord implémenté un modèle d'éclairage de Phong avec composantes ambiante, diffuse et spéculaire. Ensuite, nous ajoutons la logique pour appliquer l'ombre :

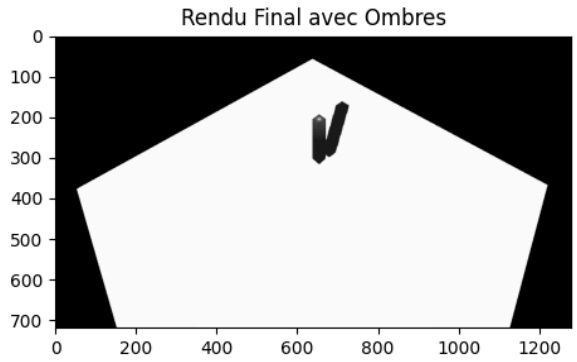
1. **Récupération de la position du fragment dans l'espace lumière**
   Cette position a été transmise depuis le vertex shader et est stockée dans le vecteur *lightPos*. Il s'agit de la position du fragment vue depuis la lumière, avant division perspective.

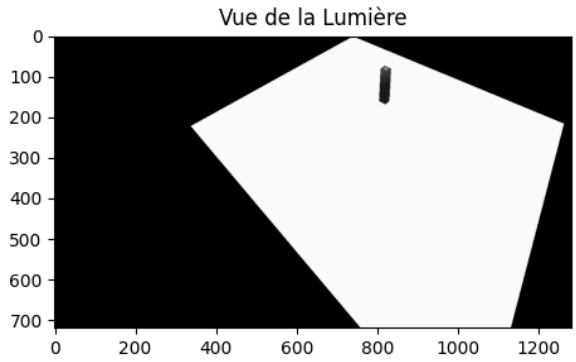
2. **Conversion en coordonnées écran pour accéder à la shadow map**
   Après division par *w*, on obtient les coordonnées normalisées. On les convertit ensuite en coordonnées d'image (entre 0 et la taille de l'image) pour accéder à la valeur de profondeur correspondante dans la shadow map.

3. **Comparaison des profondeurs avec un biais**
   On compare la profondeur du fragment actuel à celle stockée dans la shadow map. Si la profondeur actuelle est plus grande, cela signifie qu'un objet le bloque, on le considère comme dans l'ombre. Un biais est ajouté à la comparaison pour éviter les erreurs visuelles dues à l'imprécision numérique (*shadow acne*).

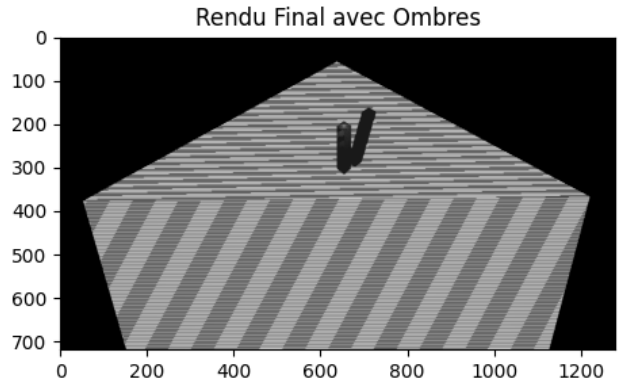
4. **Réduction de l'éclairage en cas d'ombre**
   Si le fragment est déterminé comme étant dans l'ombre, seules les composantes ambiantes sont conservées dans le calcul final, ce qui rend le pixel plus sombre.

Résultat et Performance

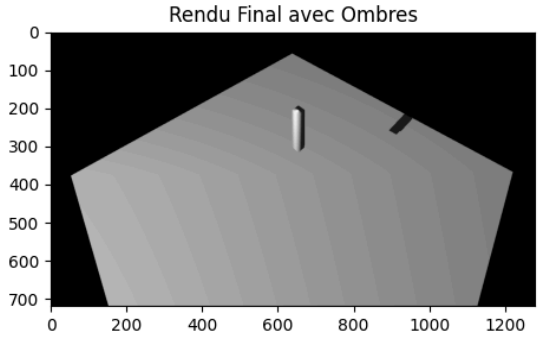


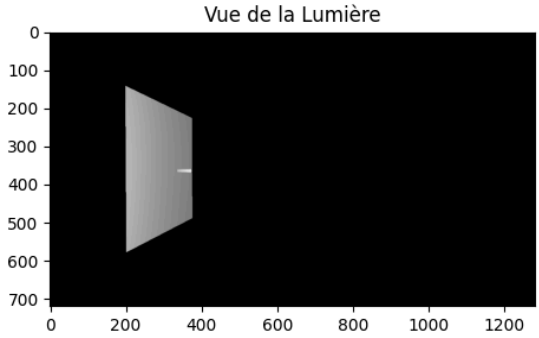


Cette méthode permet de visualiser les ombres ce qui n'était pas possible dans le code de départ. Il est important de noter la nécessité du biais car sinon on verra le shadow acne qui s'introduit à cause des erreurs de calculs.



Cependant, notre implémentation présente encore des limitations : on observe un problème au niveau du calcul de la position de l'ombre, comme le montre clairement la figure ci-dessous :





Ce problème se manifeste principalement lorsque la position de la caméra et celle de la lumière diffèrent, ce qui indique un dysfonctionnement au niveau du *fragment shader*. Nous avons tenté d'identifier l'origine du bug en analysant les différentes étapes du pipeline graphique, telles que

l'interpolation des coordonnées, le calcul de l'indice de pixel ou encore la transformation par la matrice de projection. Cependant, la cause exacte n'a pas pu être localisée dans les délais impartis.

Le *shadow mapping* introduit une étape supplémentaire de rasterisation pour chaque source de lumière présente dans la scène. Par conséquent, la complexité de l'algorithme est **linéaire** par rapport au nombre de sources lumineuses : chaque lumière nécessite de générer sa propre carte d'ombres (*shadow map*). Cela entraîne une augmentation notable du temps de calcul à mesure que le nombre de lumières augmente.
Cependant, un avantage important est que la génération des *shadow maps* est indépendante de la caméra : une fois calculées, ces cartes peuvent être utilisées pour toutes les vues, ce qui est particulièrement utile dans les scènes multi-caméras. De plus, cette étape repose sur la rasterisation, une opération hautement optimisée sur les cartes graphiques modernes, ce qui permet de limiter l'impact sur les performances globales.

Limitations

Bien que le shadow mapping soit une technique populaire pour le rendu d'ombres en 3D, elle présente plusieurs limitations :

1. **Pas d'ombres douces**
   Le shadow mapping de base produit des ombres avec des bords nets, ce qui ne correspond pas au comportement naturel des ombres dans la réalité, sauf en cas de lumière directionnelle très intense. Pour remédier à cela, des méthodes comme le *Percentage Closer Filtering (PCF)* permettent de lisser les bords, tandis que des approches plus avancées comme les *Screen-Space Shadows* ou les *Ray Traced Shadows* offrent un rendu beaucoup plus réaliste.

2. **Résolution limitée**
   La shadow map est une image 2D de profondeur. Sa résolution fixe peut poser problème, surtout dans des scènes vastes. Les ombres deviennent floues ou pixelisées, surtout à distance. Ce phénomène, connu sous le nom d'aliasing, peut être réduit grâce à des techniques telles que le *Cascaded Shadow Mapping (CSM)*, qui génère plusieurs shadow maps à différentes résolutions en fonction de la distance à la caméra.

3. **Shadow acne**
   Ce sont des artefacts visuels où un objet s'auto-projette une ombre sur lui-même, à cause d'erreurs de précision dans les comparaisons de profondeur. Pour limiter cet effet, il est courant d'introduire un biais lors de la comparaison, ce qui permet d'éviter ces auto-ombres indésirables.
   Un biais statique, appliqué de manière uniforme, peut suffire pour des surfaces planes, mais devient problématique sur des surfaces inclinées ou complexes : s'il est trop faible,

du *shadow acne* persiste ; s'il est trop fort, il provoque des artefacts de décollement où les ombres semblent flotter au-dessus des objets (*Peter-Panning*).
Plusieurs articles de recherche ont étudié ces limites du biais statique et proposé des solutions plus robustes. Par exemple, **Everitt** [Everitt02] introduit le **slope-scaled depth bias**, qui adapte dynamiquement le biais en fonction de la pente de la surface, réduisant efficacement le *shadow acne* tout en limitant les artefacts de flottaison.

Références

Shadow Mapping
https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

Shadow Maps Techniques Compared
https://www.diva-portal.org/smash/get/diva2:1677723/FULLTEXT02

Real-Time Shadows
https://www.google.fr/books/edition/Real_Time_Shadows/HfbRBQAAQBAJ?hl=en&gbpv=0

Percentage Closer Filtering (PCF)
https://developer.nvidia.com/gpugems/gpugems/part-ii-lighting-and-shadows/chapter-11-shadow-map-antialiasing

Screen-Space Shadows
https://panoskarabelas.com/posts/screen_space_shadows/

Ray Traced Shadows
https://medium.com/@alexander.wester/ray-tracing-soft-shadows-in-real-time-a53b836d123b

Cascaded Shadow Mapping (CSM)
https://learnopengl.com/Guest-Articles/2021/CSM