Rapport du projet Post-Processing : SSAO et Bloom

Logiciel et Matériel Graphique

Elise Moebs

20 janvier 2017

I. SSAO – Screen Space Ambiant Occlusion

a. Principe

La méthode SSAO permet d'approximer l'occlusion ambiante. Il s'agit des zones d'ombre dues au relief sur un objet, par exemple dans les coins d'une pièce ou dans les plis de vêtements. En effet, en rebondissant, la lumière a moins de chances d'arriver dans ces zones peu accessibles…

Grâce à la méthode SSAO de Crytek, il est possible de générer en temps réel les zones d'ombre correspondants à l'occlusion ambiante. Le principe est simple : pour chaque pixel dans la scène, on compare sa profondeur inscrite dans le depth buffer avec celle de son voisinage dans une sphère de centre le pixel. On détermine si les différents points du voisinage sont devant ou derrière la surface et on génère un occlusion buffer en faisant une moyenne des distances entre la profondeur des points du voisinage derrière la surface et la profondeur associée dans le depth buffer.

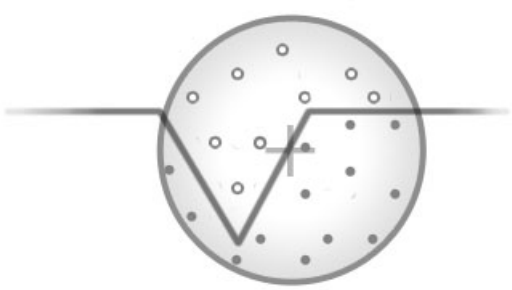


Schéma 1: Calcul de l'occlusion ambiante en fonction de la distance entre les points noirs et le pixel courant (croix

La formule de l'occlusion est la suivante :

$$Occlusion = max(0.0 , dot(N, V)) * (1.0/(1.0 + d))$$

d = distance entre le pixel courant et les échantillons occlus
N = normale associée au pixel courant
V = vecteur entre l'échantillon occlus et le pixel courant

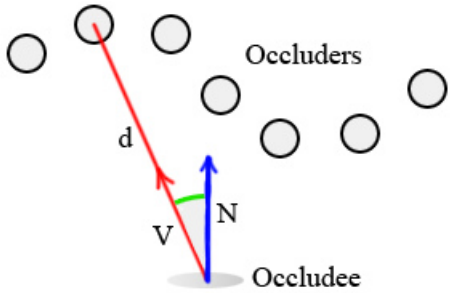


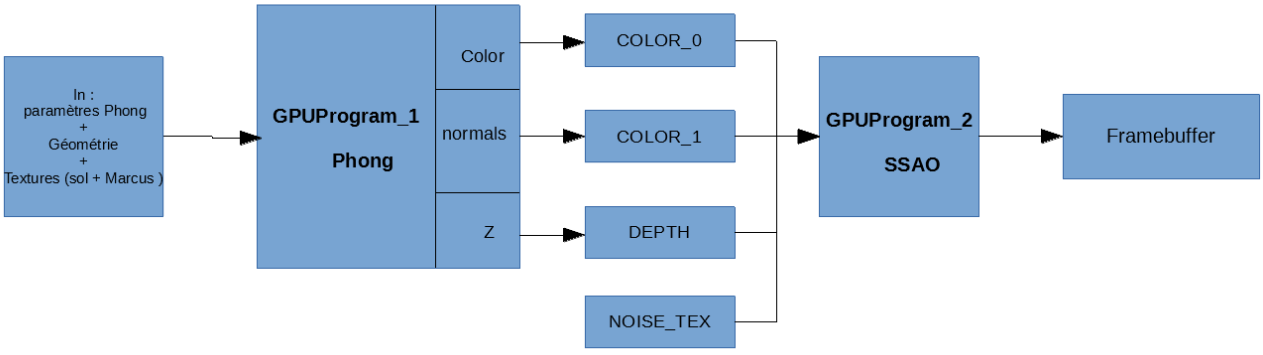
Schéma 2: N et V étant au préalable normalisés, en faisant dot(N,V) on obtient l'angle entre les deux vecteurs

Il suffit ensuite d'appliquer la formule ci-dessus à chaque échantillon et de faire une moyenne du résultat.

b. Implémentation

Entrons dans les détails de l'implémentation. Nous allons avoir besoin d'une **depth map**, d'une **normal map** et d'une **color map** contenant le rendu de la scène avec un simple Phong. Nous allons aussi avoir besoin d'une texture aléatoire (**random map**)contenant des normales aléatoires pour générer les **échantillons** aléatoires dans le voisinage du pixel courant.

On va appliquer un traitement en deux passes selon le schéma suivant :



Dans une première passe, on va générer toutes les textures dont on aura besoin pour le calcul d'occlusion ambiante, c'est-à-dire une depth map, une normal map et une color map.

Dans une deuxième passe, on utilise 4 échantillons : < 1,0 >,< -1,0 >,< 0, 1>,< 0,-1 > que l'on réfléchit avec une normale tirée de notre texture aléatoire pour rendre les échantillons aléatoires. Rendre les échantillons aléatoires est une étape clé dans l'algorithme cela permet d'éviter d'avoir des strates dans le rendu de l'occlusion ambiante.

c. Résultat

En appliquant l'agorithme ci-dessus on obtient les résultats suivants :

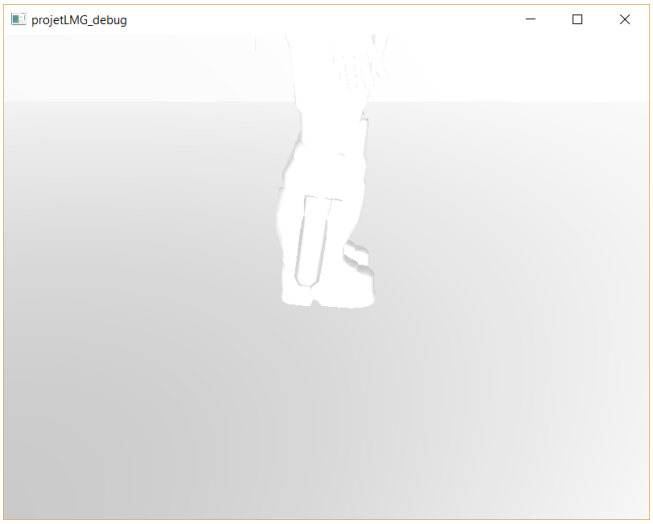


Schéma 3: Résultat de l'occlusion ambiante

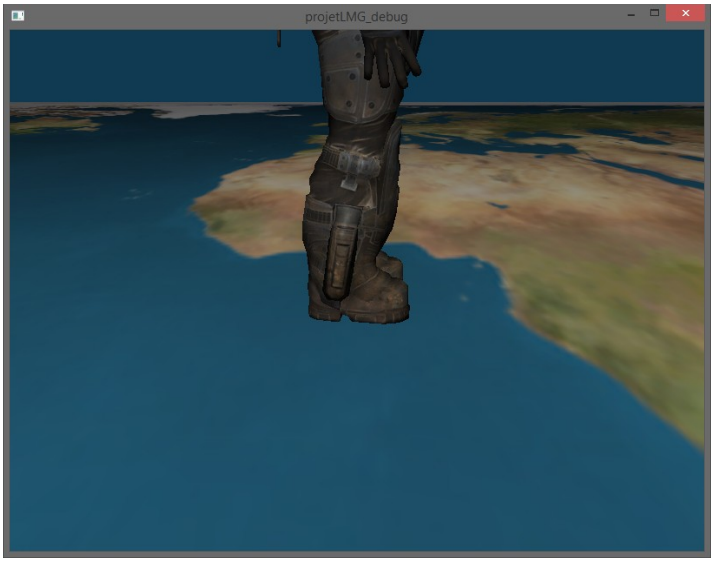


Schéma 5: Color map

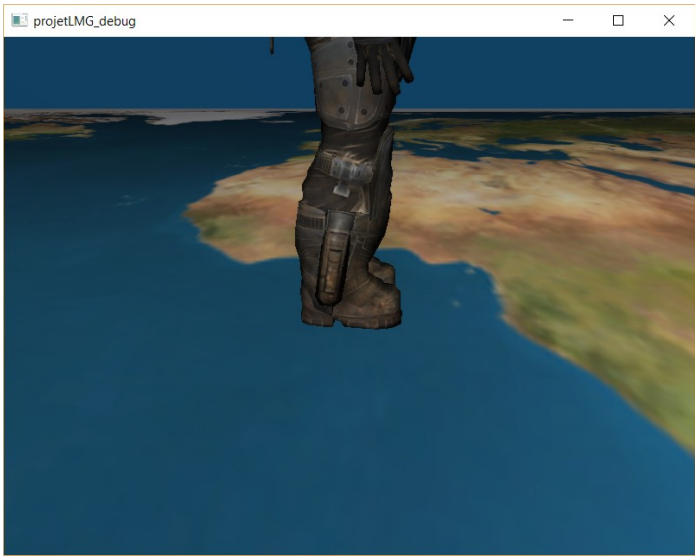


Schéma 4: Color map avec SSAO

d. Autre piste

Dans un première tentative d'implémentation, je suis partie sur un autre algorithme consistant à générer aussi des échantillons dans la voisinage du fragment courant cette fois-ci non pas dans une sphère autour mais directement sur la surface. Ils sont générés selon le code suivant :

```
in vec2 vs_texCoords;
uniform sampler2D u_texDepth;

float depth_echantillon = texture2D(u_texDepth, vec2(vs_texCoords+offset)).r;
```

En vérité, seule la profondeur de l'échantillon nous intéresse. On calcule ensuite la différence entre la profondeur de l'échantillon et celle du fragment courant. En faisant appliquant cette différence sur chaque échantillon, on peut avoir une idée de la géométrie autour du fragment courant en étudiant le signe du résultat. En faisant une moyenne des résultats on obtient le résultat suivant :

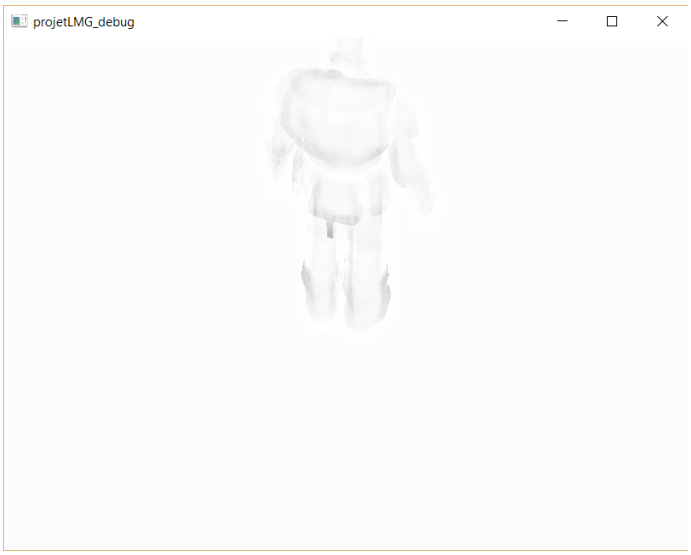


Schéma 6: Calcul de l'occlusion ambiante

Schéma 7: Calcul de l'occlusion ambiante (2)

Le résultat n'est pas encore assez satisfaisant. Pour l'améliorer un peu, on pourra par exemple choisir les échantillons de manière aléatoire. (ici ils sont choisis en carré autour du fragment courant)

II. Bloom

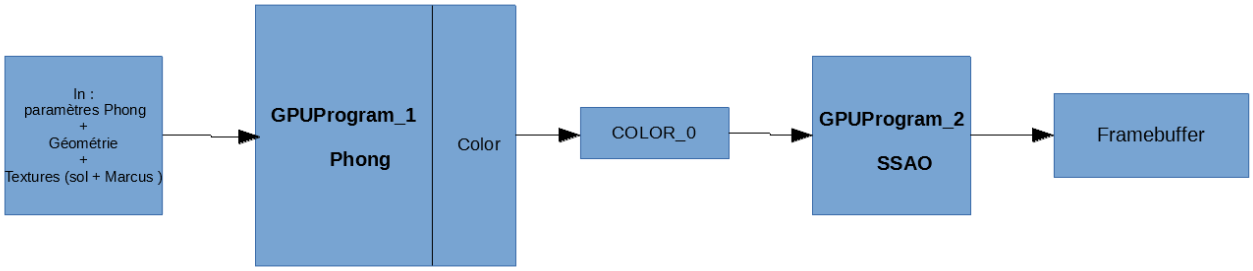
a. Principe

Effet bien moins compliqué que le SSAO, le Bloom consiste à accentuer les zones claires dans une scène en les rendant plus claires encore et en les faisant déborder sur le reste de la scène. On l'utilise si l'on cherche à reproduire par exemple un coucher de soleil ou bien des objets brillants.

Pour créer cet effet on va avoir besoin d'une **color map**. L'algorithme utilisé est le suivant : on parcourt la color map et on calcule un blur que l'on appliquera plus ou moins fort selon les zones plus ou moins claires.

b. Implémentation

Pour l'implémentation, on va partir sur un modèle semblable au SSAO toujours en deux passes :



Cette fois, on a juste besoin d'une texture contenant le rendu de la scène avec un éclairage Phong. Pour calculer le blur et l'intensité on somme les couleurs sur les fragments voisins et on applique un coefficient.

```
for( i= -4 ;i < 4; i++)
{
     for (j = -3; j < 3; j++)
     {
         sum += texture2D(u_texBlit, texcoord + vec2(j, i)*0.0004) * 0.25;
     }
}
```

On utilisera ensuite cette variable *sum* avec différents coefficients selon la couleur du fragment courant.

c. Résultat

Le résultat obtenu est le suivant : on voit bien sur l'Afrique que l'éclairage est accentué.

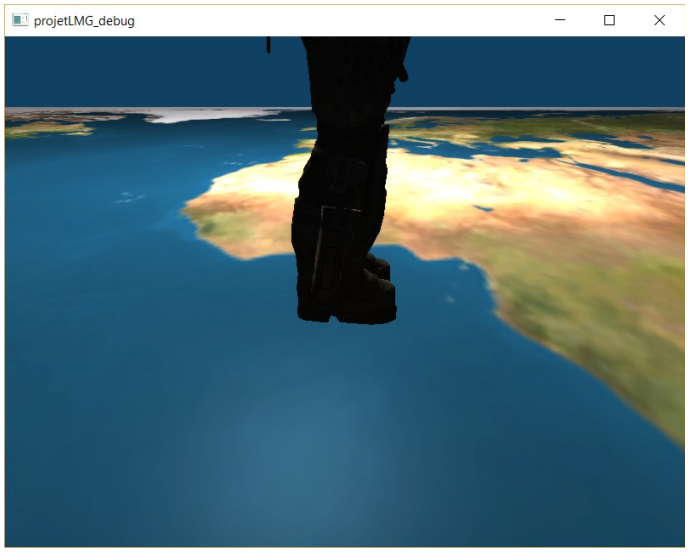


Schéma 8: Effet Bloom

III. Sources

SSAO :

https://www.gamedev.net/resources/_/technical/graphics-programming-and-theory/a-simple-and-practical-approach-to-ssao-r2753

http://john-chapman-graphics.blogspot.fr/2013/01/ssao-tutorial.html

http://theorangeduck.com/page/pure-depth-ssao

Bloom :

http://wp.applesandoranges.eu/?p=14

http://prideout.net/archive/bloom