

Synthèse d'images avancée

E. Bonnefoy & D. Delallée

March 28, 2013

1 Compilation et Exécution

Ouvrir le terminal, se placer dans le répertoire nommé "SyntheseAvanceeProjet" puis taper la commande suivante : make config=release

Pour exécuter notre programme il suffit ensuite de taper la ligne suivante :
./bin/release/projet/syntheseImage.

NB : il se peut que des erreurs apparaissent, cela vient du loader de fichiers .ply que nous utilisons pour afficher des objets dans notre scène qui n'accepte que le chemin en dur des fichiers. Afin de résoudre les erreurs, il faut aller dans le fichier main.cpp (ligne 278 à 312.) et modifier le chemin des fichiers .ply.

2 Détails de l'interface utilisateur

Pour afficher les différents mode d'affichage, nous utilisons les touches F1 à F9 :

F1 : affichage de la scène avec illumination globale

F2 : affichage de la composante matériel (ou couleur)

F3 : affichage des normales

F4 : affichage du buffer de profondeur

F5 : affichage des positions

F6 : affichage de la scene avec une illumination multi passes

F8 : affichage de la scène avec SSAO + effet de bloom

F9 : affichage de la scène avec illumination globale et SSAO

Afin de mettre en évidence l'ambient occlusion que nous avons réalisé, nous avons mis l'objet dragon dans une scène à part. On peut le visualiser grâce aux touches suivantes :

Inser : Dragon sans SSAO

Suppr : Dragon avec SSAO

Pour se déplacer dans la scène seule l'utilisation de la souris est nécessaire :

clic gauche : rotation de la caméra

clic molette : déplacement repère camera

clic droit : zoom avant/arrière sur la scène

Un menu d'aide est également disponible en appuyant sur les touches majuscules (*Caps lock* pour le faire apparaître) et shift (*left_shift* pour le faire disparaître).

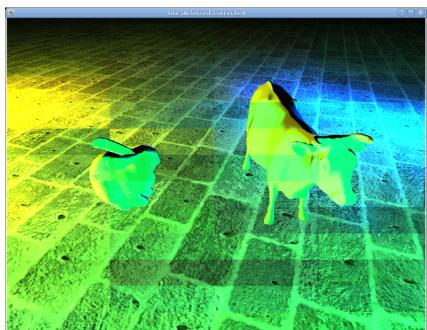
3 Explication de la scène

Notre scène est composée de treize objets : quatre arbres (4806 facettes), une aigrette (1527 facettes), un escargot (23475 facettes), un lapin (1850 facettes), une vache (5804 facettes), un lampadaire (8828 facettes), un hélicoptère (2094 facettes), une porsche (10474 facettes), un pickup (13402 facettes) et une moto atc (13594 facettes).

Au total notre scène est donc composée de 100272 facettes.

Le dragon possède, quant à lui, 871414 facettes.

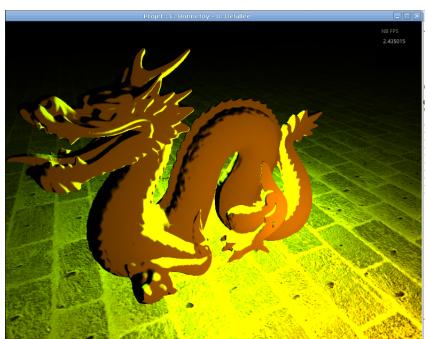
Concernant notre scène avec plusieurs objets, voici quelques photos :



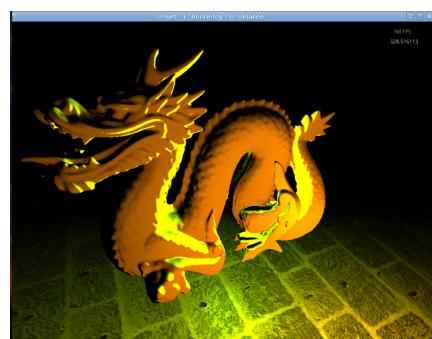
Scène sans SSAO



Scène avec SSAO

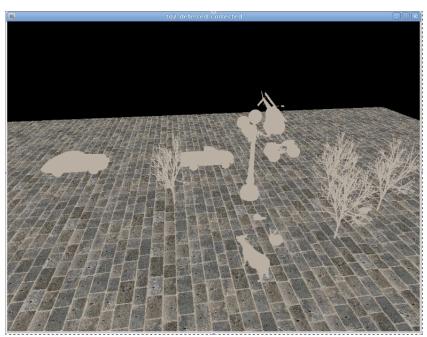


Dragon sans SSAO

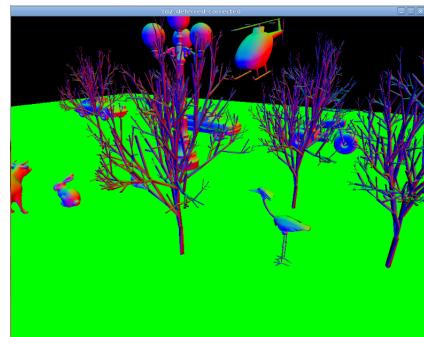


Dragon avec SSAO

Pour finir, nous mettrons quelques photos des couleurs, normales, profondeurs qui nous ont permis de réaliser la SSAO.



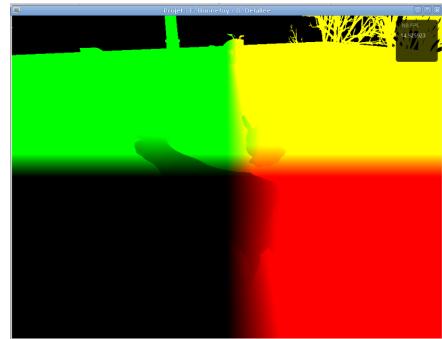
Couleur de la scène



Normales de la scène



Profondeur de la scène



Position de la scène

Les ombres rectangles que l'on peut apercevoir sur la première image sont des artefacts dus aux changements de lumières (nous avons fait légèrement scintiller nos lumières au cours du temps) durant l'impression d'écran (qui prend environs une dizaine de secondes).

4 Explication de notre travail

4.1 Illumination globale en Deferred Shading

Toute l'illumination globale de notre scène est réalisé en deferred shading en deux passes. Après création des différents objets, un premier shader (gbuffer_shader) se charge de placer dans des textures les informations de la scène: les couleurs, les normales, la profondeur et la position.

Un deuxième shader (laccum_shader, pour light accumulation) récupère ces données et calcule l'illumination de la scène. Il affiche ensuite le résultat sous forme de texture qu'il plaque sur un quad de la taille de notre fenêtre OpenGL.

4.2 Ambient occlusion

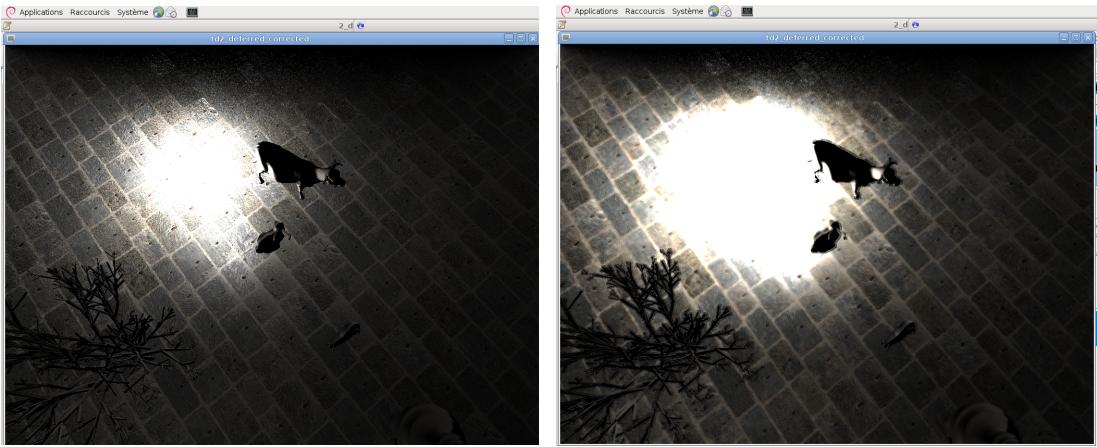
Par la suite, nous avons implémenté l'ambient occlusion. Nous utilisons toujours les données textures fournies par le shader gbuffer, mais nous modifions légèrement le shader laccum pour lui ajouter le calcul de l'ambient occlusion, qui consiste en regarder pour un point donnée ainsi que ses voisins leur distance et l'illumination de la scène afin de savoir quelle partie doivent être plus ou moins sombre

Afin de connaître l'efficacité de notre scène, nous avons calculé le nombre de FPS. Nous calculons le temps d'exécution pour un cycle de rendu, puis nous divisons 1.0 par le temps obtenu. Le nombre de FPS s'affiche dans un cadre en haut à gauche de notre écran.

Le résultat visuel de l'ambient occlusion est satisfaisant, mais le nombre de FPS chute de beaucoup.

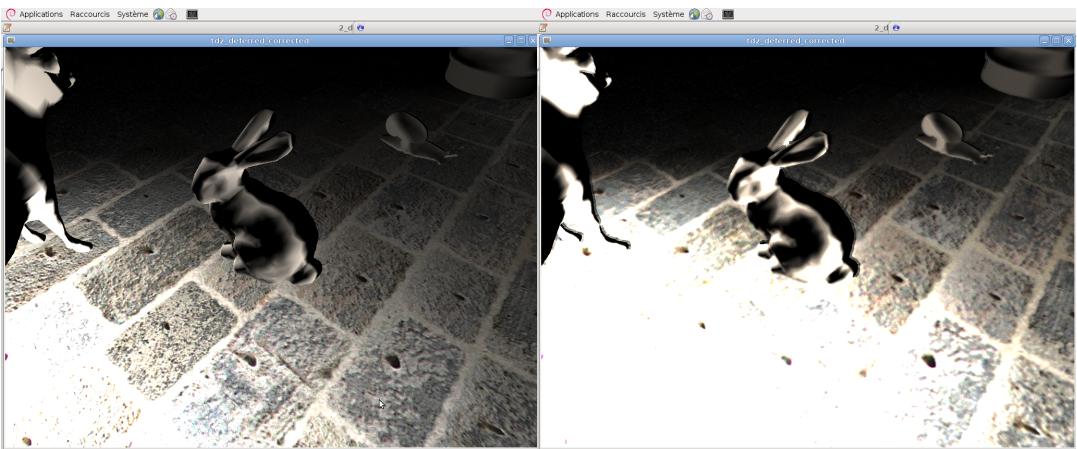
4.3 Effet Bloom en render-to-texture

Nous avons ensuite implémenté l'effet Bloom. Nous avons créé pour cela un nouveau shader (bloomTexture) qui récupère dans une texture la sortie du shader laccum gérant l'illumination globale, puis calcule l'effet bloom sur cette image (effet de flou sur une fenêtre de pixels de taille 3x4). On affiche alors sa sortie en la plaquant sur un quad. Ci-dessous quelques impressions d'écran:



Scène sans bloom

Scène avec bloom



Scène sans bloom

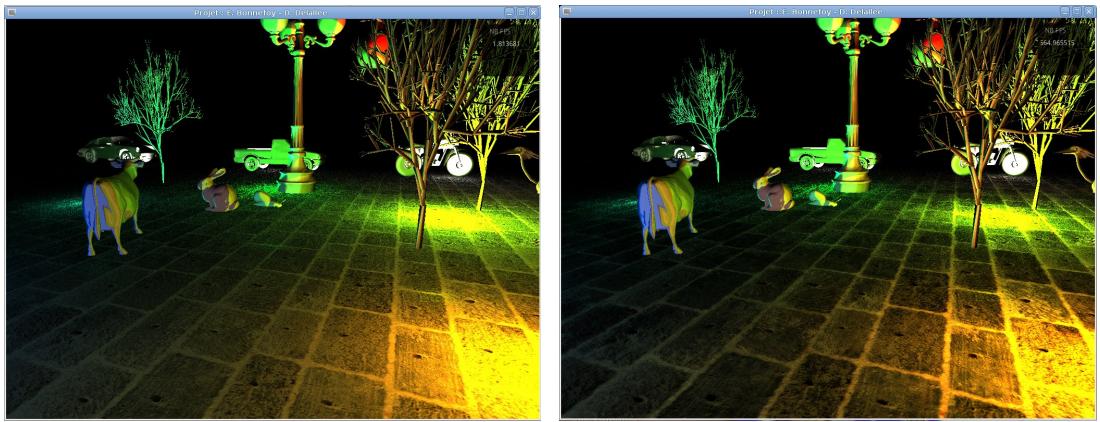
Scène avec bloom

Nous avons néanmoins quelques artefacts qui apparaissent avec l'effet bloom, dus au calcul de moyenne sur la fenêtre de pixels servant à flouter le les contours des objets.

Par ailleurs, nous nous sommes heurtés au problème suivant: notre illumination globale, calculée par le shader laccum, rafraîchit l'affichage du quad de rendu à chaque ajout d'une nouvelle source de lumière. L'affichage de la scène se fait correctement à la sortie du shader, mais la récupération de la sortie de ce shader en tant que texture nous rendait uniquement la dernière lumière traitée par laccum. Notre effet bloom ne peut donc être calculé qu'avec une seule source de lumière (ici une lumière blanche, et de forte intensité pour visualiser l'effet du bloom).

4.4 Rehausseur de contraste

Ce 'rehausseur de contraste' est en fait le premier essai d'effet bloom que nous avons testé, sans passer par une passe de rendu-to-texture. En effet, notre première idée a été de ne pas recréer un shader supplémentaire pour l'effet bloom, mais de modifier le shader laccum afin de lui ajouter le calcul du bloom. Nous ne pouvions alors pas calculer l'effet sur la scène finale illuminée (qui est la sortie du laccum), nous avons donc tenté de le calculer sur la texture des couleurs de la scène. L'effet obtenu est très loin de l'effet bloom, mais rehausse légèrement les contrastes de notre scène. Su les screenshots ci-dessous, l'effet est notamment visible sur les pavés qui constituent le sol de notre scène.

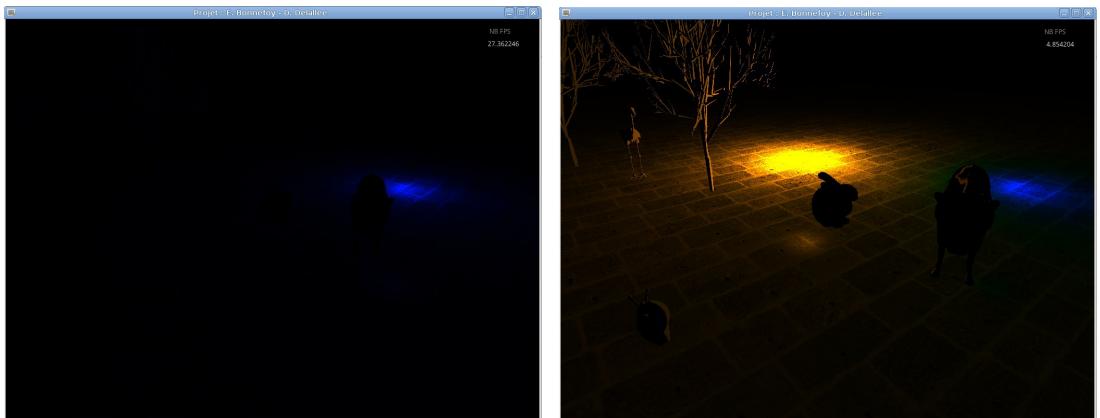


Scène normale

Scène au contraste rehaussé

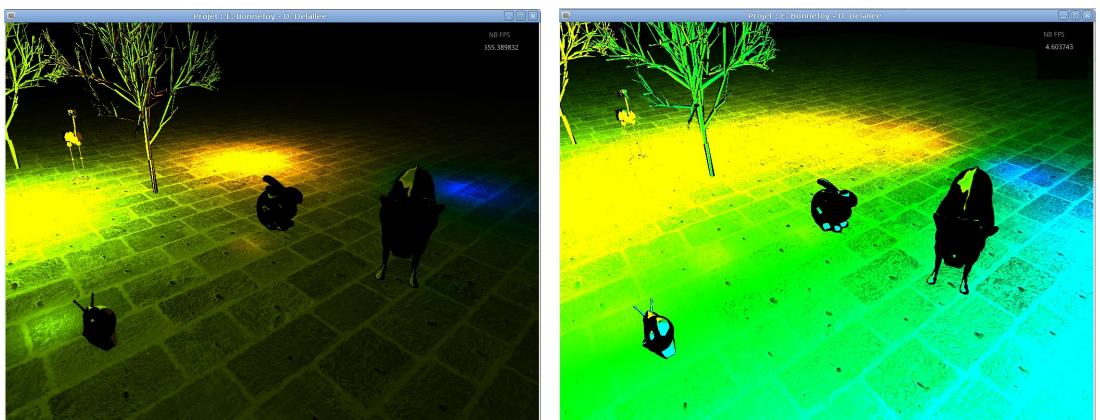
4.5 Illumination globale en multi-passe

Pour pallier à l'inconvénient du laccum shader (qui réaffiche le quad à chaque nouvelle source de lumière), nous avons tenté d'implémenter l'illumination globale en multi-différé. Nous appelons le shader laccum autant de fois qu'il y a de source de lumières dans la scène, chaque ajout de lumière dans la scène étant calculé sur la texture de rendu de la lumière précédente. Nous appelons donc le shader laccum au sein d'une boucle (un petit shader de transfert se charge de récupérer la sortie de laccum et de lui réenvoyer à chaque itération), puis nous affichons à la fin de la boucle la texture finale sur un quad. Voici les résultats pour 1, 2, 3 et 12 sources de lumières:



Une source de lumière

Deux sources de lumières



Trois sources de lumières

Douze sources de lumières

On remarque que l'illumination se comporte étrangement sur les objets (très visible sur l'image avec 12 sources de lumières). Nous pensons qu'il s'agit de la formule d'illumination que nous avons utilisé, mais nous nous en sommes rendus compte à un moment où nous n'avions plus le temps d'en utiliser une autre. Un travail futur à effectuer serait de tester une autre formule d'illumination globale est vérifier si les différents artefacts rencontrés lors de l'effet bloom et l'illumination multi-passes disparaissent.