OpenGL

Tampons de profondeur et de pochoir

Par Alexander Overvoorde - Alexandre Laurent (traducteur)

Date de publication : 13 septembre 2015

Navigation....3
I - Tampons supplémentaires....3
II - Préparations....3
III - Tampon de profondeur....4
IV - Tampon de pochoir....4
IV-A - Définir les valeurs....5
IV-B - Utiliser les valeurs dans les opérations de dessin....6
V - Réflexions sur un plan....7
VI - Exercices....8
VII - Remerciements....9
Navigation....9

Le contenu de cet article est rédigé par Alexander Overvoorde et est mis à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 3.0 non transposé. Les logos Developpez.com, en-tête, pied de page, css, et look & feel de l'article sont Copyright ® 2013 Developpez.com.

Navigation

Le contenu dynamique ne peut pas être affiché dans ce support, veuillez consulter la page **en ligne** pour le visualiser.

Tutoriel précédent : **transformat**

Sommaire

Tutoriel suivant : **tampons de rendu**

I - Tampons supplémentaires

Jusqu'à présent, nous n'avons utilisé qu'un seul type de tampon : le tampon de couleurs. Ce chapitre portera sur deux autres types de tampons, le tampon de profondeur et le tampon pochoir (« stencil buffer »). Pour chacun d'eux, un problème sera présenté et ensuite résolu en utilisant ce tampon.

II - Préparations

La meilleure méthode pour démontrer l'utilisation de ces tampons est de dessiner un cube au lieu d'une géométrie plate. Le vertex shader doit être modifié pour accepter une troisième coordonnée :

```
in vec3 position;
...
gl_Position = proj * view * model * vec4(position, 1.0);
```

Nous allons aussi avoir besoin d'altérer les couleurs dans la suite de ce chapitre, donc assurez-vous que le fragment shader multiplie la couleur de la texture avec la couleur de l'attribut :

```
vec4 texColor = mix(texture(texKitten, Texcoord),
                    texture(texPuppy, Texcoord), 0.5);
outColor = vec4(Color, 1.0) * texColor;
```

Maintenant les sommets sont composés de huit nombres à virgule flottante, vous devez donc mettre à jour le décalage des attributs de sommets ainsi que ceux du pas. Finalement, ajoutez les nouvelles coordonnées aux tableaux de sommets :

```
float vertices[] = {
  // X      Y     Z     R     G     B     U     V
    -0.5f,  0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
     0.5f,  0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f,
     0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f,
    -0.5f, -0.5f, 0.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f
};
```

Confirmez que vos changements sont effectifs en exécutant le programme et en vérifiant qu'il affiche bien une image rotative plate d'un petit chat mélangée à l'image d'un chiot. Un simple cube est composé de 36 sommets (6 côtés * 2 triangles * 3 sommets), donc je vais vous faciliter la vie en vous fournissant le **tableau ici**.

```
glDrawArrays(GL_TRIANGLES, 0, 36);
```

Nous n'avons pas besoin d'un tampon d'éléments pour dessiner ce cube, vous pouvez donc utiliser `glDrawArrays` pour l'afficher. Si vous êtes confus avec cette explication, vous pouvez comparer votre programme avec **ce code**.

Le contenu dynamique ne peut pas être affiché dans ce support, veuillez consulter la page **en ligne** pour le visualiser.

En voyant le résultat, il est évident que le cube n'est pas affiché comme prévu. Les côtés du cube sont dessinés, mais ils se recouvrent d'une étrange manière ! Le problème est que lorsque OpenGL dessine votre cube triangle par

Le contenu de cet article est rédigé par Alexander Overvoorde et est mis à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 3.0 non transposé. Les logos Developpez.com, en-tête, pied de page, css, et look & feel de l'article sont Copyright ® 2013 Developpez.com.
http://opengl.developpez.com/tutoriels/open_gl/depth-stencil/

triangle, il va simplement écrire sur les pixels, même s'il y avait autre chose de dessiné auparavant. Dans ce cas, OpenGL va joyeusement dessiner les triangles de l'arrière sur les triangles de l'avant.

Heureusement OpenGL offre des méthodes pour indiquer quand il faut ou non dessiner un pixel. Dans ce chapitre, je vais présenter deux des méthodes les plus courantes qui sont le test de profondeur et le test de pochoir.

III - Tampon de profondeur

Le *Z-buffering* est une technique pour garder la profondeur de chaque pixel de l'écran. La profondeur est proportionnelle à la distance entre le plan écran et le fragment qui a été dessiné. Cela signifie que les fragments sur les côtés du cube qui sont le plus loin du spectateur ont une plus haute valeur en profondeur, alors que les fragments les plus proches ont une plus petite valeur.

Si la profondeur est stockée avec la couleur lorsqu'un fragment est écrit, les fragments affichés plus tard peuvent comparer leur profondeur avec celle inscrite dans le tampon afin de savoir si le nouveau fragment est plus proche du spectateur que l'ancien fragment. Si c'est le cas, il doit être dessiné par-dessus, sinon il sera simplement oublié. Cela s'appelle le test de profondeur.

OpenGL permet de stocker ces valeurs de profondeur dans un tampon supplémentaire, appelé le tampon de profondeur et exécute automatiquement les vérifications nécessaires pour les fragments. Le fragment shader ne va pas s'exécuter pour les fragments qui sont invisibles, ce qui peut radicalement améliorer les performances. Cette fonctionnalité peut être activée en appelant la fonction glEnable (**doc**).

```
glEnable(GL_DEPTH_TEST);
```

Si vous activez cette fonctionnalité et que vous exécutez à nouveau votre application, vous allez obtenir un écran noir. Cela se produit, car le tampon de profondeur ne contient, par défaut, que des 0 pour la profondeur. Comme aucun fragment ne peut être plus proche que cette valeur, tous ont été ignorés.

Le tampon de profondeur peut être nettoyé en même temps que le tampon de couleurs en modifiant l'appel à glClear (**doc**) :

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

La valeur de réinitialisation par défaut pour la profondeur est 1.0f, qui est équivalente à la profondeur du plan lointain et donc la profondeur la plus grande possible. Tous les fragments seront plus proches que celle-ci et ne seront donc plus ignorés.

Le contenu dynamique ne peut pas être affiché dans ce support, veuillez consulter la page **en ligne** pour le visualiser.

Avec l'activation du test de profondeur, le cube est maintenant affiché correctement. Tout comme pour le tampon de couleurs, le tampon de profondeur a une certaine précision que vous pouvez spécifier. Utiliser moins de bits va réduire la consommation mémoire, mais peut augmenter le nombre d'erreurs dans les scènes complexes.

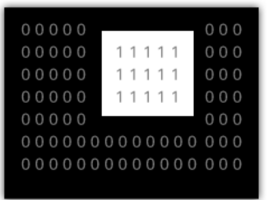
IV - Tampon de pochoir

Le tampon de pochoir est une extension optionnelle du tampon de profondeur qui donne plus de contrôle sur quels fragments seront affichés ou non. Comme le tampon de profondeur, une valeur est stockée pour chaque pixel, mais cette fois, vous avez le contrôle sur le changement de cette valeur et lorsque le fragment ne doit pas être dessiné suivant cette valeur. Notez que si votre test de profondeur échoue, le test de pochoir ne détermine pas si le fragment est dessiné ou non, mais ces fragments continuent d'affecter le tampon de pochoir !

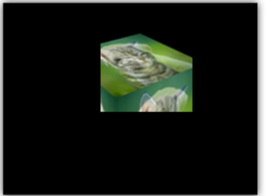
Pour se familiariser avec un tampon de pochoir avant de l'utiliser, analysons ce simple exemple.

Le contenu de cet article est rédigé par Alexander Overvoorde et est mis à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 3.0 non transposé. Les logos Developpez.com, en-tête, pied de page, css, et look & feel de l'article sont Copyright ® 2013 Developpez.com.

Color buffer without stencil test



Stencil buffer


Color buffer with stencil test

Dans ce cas, le tampon de pochoir a été réinitialisé avec des zéros et ensuite un rectangle ayant une valeur de 1 a été dessiné. L'opération de dessin du cube utilise les valeurs du tampon de pochoir pour effectuer le dessin que si la valeur de stencil est à 1.

Maintenant que vous avez compris ce que fait le tampon de pochoir, nous allons voir les fonctions OpenGL correspondantes.

```
glEnable(GL_STENCIL_TEST);
```

Le test de pochoir est activé avec un appel à la fonction glEnable (**doc**), tout comme pour le test de profondeur. Vous n'avez pas encore à ajouter cet appel dans votre code. Je vais d'abord détailler les fonctions dans les deux prochaines sections et nous allons faire une démonstration sympa.

IV-A - Définir les valeurs

Les opérations classiques de dessin sont utilisées pour déterminer quelles valeurs le tampon de pochoir sont affectées par quelle opération sur le pochoir. Si vous souhaitez définir un rectangle avec une valeur spécifique comme dans l'exemple ci-dessus, dessinez simplement un rectangle 2D dans cette zone. L'opération appliquée à ces valeurs peut être contrôlée avec les fonctions glStencilFunc (**doc**), glStencilOp (**doc**) et glStencilMask (**doc**).

La fonction glStencilFunc (**doc**) est utilisée pour spécifier les conditions déterminant si le fragment passe le test de pochoir. Voici une description de ses paramètres :

- func : la fonction de test qui peut être GL_NEVER, GL_LESS, GL_LEQUAL, GL_GREATER, GL_GEQUAL, GL_EQUAL, GL_NOTEQUAL, et GL_ALWAYS ;
- ref : une valeur à laquelle la valeur de pochoir va être comparée dans la fonction de test ;
- mask : une opération ET binaire appliquée entre le masque et la valeur du tampon de pochoir ainsi que la valeur de référence avant le test ;

Si vous ne souhaitez pas que les valeurs du tampon de pochoir inférieures à 2 soient affectées, vous utiliseriez :

```
glStencilFunc(GL_GEQUAL, 2, 0xFF);
```

La valeur du masque est entièrement composée de 1 (dans le cas d'un tampon de pochoir de 8 bits), donc il n'impactera pas le test.

La fonction glStencilOp (**doc**) indique ce qui doit être effectué sur les valeurs de pochoir suivant le résultat du test de pochoir et du test de profondeur. Les paramètres sont :

- sfail : action à effectuer si le test de pochoir échoue ;
- dpfail : action à effectuer si le test de pochoir est réussi, mais que le test de profondeur a échoué ;
- dppass : action à prendre si les deux tests ont réussi.

Les valeurs de pochoir peuvent être modifiées comme suit :

- GL_KEEP : la valeur actuelle est conservée ;

Le contenu de cet article est rédigé par Alexander Overvoorde et est mis à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 3.0 non transposé. Les logos Developpez.com, en-tête, pied de page, css, et look & feel de l'article sont Copyright ® 2013 Developpez.com.
http://opengl.developpez.com/tutoriels/open_gl/depth-stencil/

- GL_ZERO : la valeur de pochoir est mise à 0 ;
- GL_REPLACE : la valeur de pochoir est définie à la valeur de référence passée à la fonction glStencilFunc ;
- GL_INCR : la valeur de pochoir est augmentée de 1 si elle est inférieure à la valeur maximale ;
- GL_INCR_WRAP : pareil que GL_INCR sauf que la valeur repasse à 0 lorsque la valeur maximale est dépassée ;
- GL_DECR : la valeur de pochoir est diminuée de 1 si elle est supérieure à 0 ;
- GL_DECR_WRAP : pareil que GL_DECR sauf que la valeur repasse à la valeur maximale lorsque la valeur actuelle est 0 (le tampon de pochoir conserve des entiers non signés) ;
- GL_INVERT : une opération binaire d'inversion de la valeur.

Finalement, la fonction glStencilMask (**doc**) peut être utilisée pour contrôler les bits qui sont écrits dans le tampon de pochoir lorsqu'une opération est exécutée. Par défaut tout est à 1, ce qui signifie que le résultat de n'importe quelle opération n'a aucun effet.

Si, comme dans l'exemple, vous souhaitez définir toutes les valeurs d'une zone rectangulaire à 1, vous devez utiliser les appels suivants :

```
glStencilFunc(GL_ALWAYS, 1, 0xFF);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
glStencilMask(0xFF);
```

Dans ce cas, le rectangle ne doit pas être dessiné dans le tampon de couleurs, car il n'est utilisé que pour déterminer les valeurs du tampon de pochoir.

```
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glDepthMask(GL_FALSE);
```

La fonction glColorMask (**doc**) vous permet de spécifier quelles données sont écrites dans le tampon de couleurs lors de l'opération de dessin. Dans ce cas, nous souhaitons désactiver tous les canaux de couleurs (rouge, vert, bleu, alpha). L'écriture dans le tampon de profondeur doit être désactivée séparément avec la fonction glDepthMask (**doc**), afin que le dessin du cube ne soit pas impacté par une quelconque altération des valeurs de la profondeur. C'est plus propre que de simplement réinitialiser le tampon de profondeur une nouvelle fois.

IV-B - Utiliser les valeurs dans les opérations de dessin

En sachant comment définir les valeurs, leur utilisation pour tester les fragments durant les opérations de dessin devient très évidente. Tout ce que vous devez faire est de réactiver l'écriture de la couleur et de la profondeur si vous les aviez désactivées plus tôt et définir la fonction de test pour déterminer quels sont les fragments qui seront dessinés selon les données du tampon de pochoir.

```
glStencilFunc(GL_EQUAL, 1, 0xFF);
```

Si vous utilisez cet appel pour définir la fonction de test, le test ne va réussir que pour les pixels avec une valeur égale à 1. Un fragment ne sera dessiné que s'il passe les tests de profondeur et de pochoir, donc la définition de glStencilOp (**doc**) n'est pas nécessaire. Dans l'exemple ci-dessus, seules les valeurs à l'intérieur du rectangle ont été définies à 1, donc seuls les fragments du cube à l'intérieur de cette zone seront dessinés.

```
glStencilMask(0x00);
```

Un détail qu'il est facile d'oublier est que le dessin du cube peut continuer d'affecter les valeurs dans le tampon de pochoir. Ce problème peut être résolu en définissant le masque du pochoir à 0, désactivant ainsi l'écriture des valeurs de stencil.

Le contenu de cet article est rédigé par Alexander Overvoorde et est mis à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 3.0 non transposé. Les logos Developpez.com, en-tête, pied de page, css, et look & feel de l'article sont Copyright ® 2013 Developpez.com.

V - Réflexions sur un plan

Épiçons notre démo en ajoutant un sol avec une réflexion sous le cube. Je vais ajouter les sommets pour le sol dans le même tampon de sommets que celui du cube pour garder les choses simples :

```
float vertices[] = {
    ...

    -1.0f, -1.0f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
     1.0f, -1.0f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
     1.0f,  1.0f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f,
     1.0f,  1.0f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f,
    -1.0f,  1.0f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
    -1.0f, -1.0f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f
}
```

Maintenant ajoutez un appel supplémentaire dans la boucle principale :

```
glDrawArrays(GL_TRIANGLES, 36, 6);
```

Pour créer une réflexion du cube, il suffit de l'afficher une nouvelle fois avec l'axe des Z inversé :

```
model = glm::scale(
    glm::translate(model, glm::vec3(0, 0, -1)),
    glm::vec3(1, 1, -1)
);
glUniformMatrix4fv(uniModel, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

J'ai défini la couleur des sommets du sol en noir afin que le sol n'affiche pas la texture, donc si vous changez la couleur de nettoyage à blanc, vous serez capable de le voir. J'ai aussi changé les paramètres de la caméra pour avoir un meilleur aperçu de la scène.

Le contenu dynamique ne peut pas être affiché dans ce support, veuillez consulter la page **en ligne** pour le visualiser.

Deux problèmes apparaissent sur l'image affichée :

- le sol cache la réflexion à cause du test de profondeur ;
- la réflexion est visible en dehors du sol.

Le premier problème est simple à corriger en désactivant temporairement l'écriture dans le tampon de profondeur lors de l'affichage du sol :

```
glDepthMask(GL_FALSE);
glDrawArrays(GL_TRIANGLES, 36, 6);
glDepthMask(GL_TRUE);
```

Pour corriger le second problème, il est nécessaire d'ignorer les fragments qui sont hors du sol. Il est donc temps de voir ce à quoi sert le test de pochoir.

Il peut être actuellement grandement bénéfique de créer une liste des étapes du rendu de la scène pour obtenir une idée correcte de ce qui se passe.

- dessiner un cube classique ;
- activer le test de pochoir et définir la fonction de test et les opérations pour écrire des 1 pour tous les pixels voulus ;
- afficher le sol ;
- définir la fonction de pochoir pour réussir le test lorsque la valeur du pochoir est à 1 ;
- afficher le cube inversé ;

Le contenu de cet article est rédigé par Alexander Overvoorde et est mis à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 3.0 non transposé. Les logos Developpez.com, en-tête, pied de page, css, et look & feel de l'article sont Copyright ® 2013 Developpez.com.

- désactiver le test pochoir.

Le nouveau code de rendu ressemble à ceci :

```
glEnable(GL_STENCIL_TEST);

    // Afficher le sol
    glStencilFunc(GL_ALWAYS, 1, 0xFF); // Définir n'importe quelle valeur de pochoir à 1
    glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
    glStencilMask(0xFF); // Écrire le tampon de pochoir
    glDepthMask(GL_FALSE); // Ne pas écrire le tampon de profondeur
    glClear(GL_STENCIL_BUFFER_BIT); // Réinitialisation du tampon de pochoir (0 par défaut)

    glDrawArrays(GL_TRIANGLES, 36, 6);

    // Afficher la réflexion du cube
    glStencilFunc(GL_EQUAL, 1, 0xFF); // Réussit le test si la valeur de pochoir est 1
    glStencilMask(0x00); // Ne pas écrire dans le tampon de pochoir
    glDepthMask(GL_TRUE); // Écrire le tampon de profondeur

    model = glm::scale(
        glm::translate(model, glm::vec3(0, 0, -1)),
        glm::vec3(1, 1, -1)
    );
    glUniformMatrix4fv(uniModel, 1, GL_FALSE, glm::value_ptr(model));
    glDrawArrays(GL_TRIANGLES, 0, 36);

glDisable(GL_STENCIL_TEST);
```

J'ai annoté le code ci-dessus avec des commentaires, mais les étapes doivent être claires pour la section du tampon de pochoir.

Il ne reste plus qu'une touche finale, pour noircir un peu la réflexion cube afin de rendre la réflexion du sol moins proche d'un miroir parfait. J'ai choisi de créer une variable uniforme appelée overrideColor dans le vertex shader :

```
uniform vec3 overrideColor;
...
Color = overrideColor * color;
```

Et le code d'affichage de la réflexion du cube

```
glUniform3f(uniColor, 0.3f, 0.3f, 0.3f);
    glDrawArrays(GL_TRIANGLES, 0, 36);
glUniform3f(uniColor, 1.0f, 1.0f, 1.0f);
```

Où uniColor est la valeur de retour de l'appel glGetUniformLocation.

Le contenu dynamique ne peut pas être affiché dans ce support, veuillez consulter la page **en ligne** pour le visualiser.

Génial ! J'espère, surtout dans les chapitres comme celui-ci, que vous trouvez que travailler avec une bibliothèque bas niveau telle qu'OpenGL peut être amusant et apporter des défis intéressants ! Comme toujours, le code final est **disponible ici**.

VI - Exercices

Il n'y a pas vraiment d'exercice pour ce chapitre, mais il y a beaucoup plus d'effets intéressants que vous pouvez créer avec le tampon de pochoir. Je vous laisse chercher l'implémentation d'autres effets, tels que les **ombres avec pochoir** et le **contour d'un objet**.

Le contenu de cet article est rédigé par Alexander Overvoorde et est mis à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 3.0 non transposé. Les logos Developpez.com, en-tête, pied de page, css, et look & feel de l'article sont Copyright ® 2013 Developpez.com.

VII - Remerciements

Cet article est une traduction autorisée dont le texte original peut être trouvé sur **open.gl**.

Navigation

Tutoriel précédent : **transformat**	**Sommaire**	Tutoriel suivant : **tampons de rendu**

Le contenu de cet article est rédigé par Alexander Overvoorde et est mis à disposition selon les termes de la Licence Creative Commons Attribution - Partage dans les Mêmes Conditions 3.0 non transposé. Les logos Developpez.com, en-tête, pied de page, css, et look & feel de l'article sont Copyright ® 2013 Developpez.com.
http://opengl.developpez.com/tutoriels/open_gl/depth-stencil/