

Workshop Graphique

Raphaël *Shugo* BOISSEL (boisse_r)



Vendredi 23 Novembre 2012

Table des matières

I	Introduction	3
1	Fonctionnement du workshop	3
2	Le choix des armes	3
3	Les Wrappers DirectX et OpenGL	3
II	L'interface	5
1	Afficher une fenetre	5
1.1	objectifs	5
1.2	conseils	5
2	Dessiner un arriere plan	6
2.1	objectifs	6
2.2	conseils	6
3	Dessiner des sprites	7
3.1	objectifs	7
3.2	conseils	7
4	Un peu de logique	9
4.1	objectifs	9
4.2	conseils	9
III	Et maintenant de la 3D	10
1	Mon vaisseau c'est un cube!	10
1.1	objectifs	10
1.2	conseils	10
2	Des particules	11
2.1	objectifs	11
2.2	conseils	11
3	Charger de vrai objets	12
3.1	objectifs	12
3.2	conseils	12
4	Un peu de mouvement	13
4.1	objectifs	13
4.2	conseils	13
IV	Shaders	14
1	Lumière	14
1.1	objectifs	14
2	Ombrage celluloide	15

2.1	objectifs	15
2.2	conseils	15
V	Conclusion	16

I Introduction

L'objectif de ce workshop est la réalisation d'un petit jeu dans lequel nous nous focaliserons principalement sur tout les problème de rendu graphique. Le jeux qui vas être réalisé est un pseudo R-Type.

1 Fonctionnement du workshop

Le workshop est divisé en palier successif (du plus facile au plus difficile). Chaque palier est divisé en deux parties **LISEZ BIEN LES DEUX PARTIES AVANT DE COMMENCER LE PALIER**. Une premiere partie décrit les objectifs du palier, une deuxieme partie donne des indications sur la façons de le réaliser. Afin d'éviter que ce workshop se transforme en un gros copier coller de code ce qui serait un peu enuyeux seul les grandes lignes les principales fonctions et les pieges à éviter vous seront donné le code et la structure du code est entièrement à votre charge !

2 Le choix des armes

Dans ce workshop vous êtes libre de choisir l'outils de votre choix, de préférence celui que vous utiliserez dans votre projet (XNA, DirectX, OpenGL, ou d'autres). Cependant nous vous conseillons vivement de ne pas avoir les yeux plus gros que le ventre et de chosir DirectX ou OpenGL **UNIQUEMENT** si votre projet l'exige ou si vous êtes déjà familier avec ce type d'outils.

3 Les Wrappers DirectX et OpenGL

Pour pouvoir utiliser OpenGL ou DirectX en C# il faut utiliser des wrappers. les wrappers se présentes généralement sous la forme de bibliothèques reprenant une à une les fonctions de la bibliothèque de base dans une version accessible depuis C#. Voici les wrappers que nous vous conseillons mais rien ne vous empêche de choisir le votre :

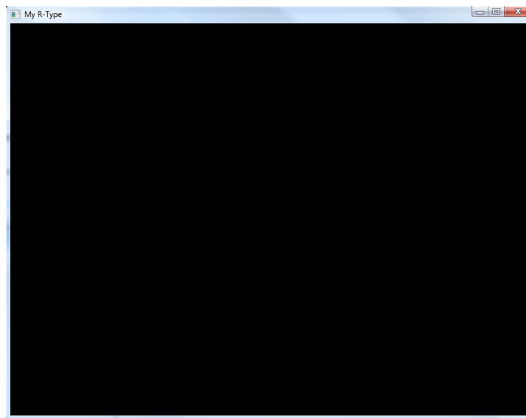
- SlimDX pour DirectX
- OpenTX pour OpenGL

NOTE : en fonction du wrapper que vous choisirez vous constaterez quelques légère différence dans le nomage des fonctions ou dans celui les paramètres : par exemple openTK utilise `Gl.Begin` et Tao `glBegin`

II L'interface

1 Afficher une fenetre

1.1 objectifs



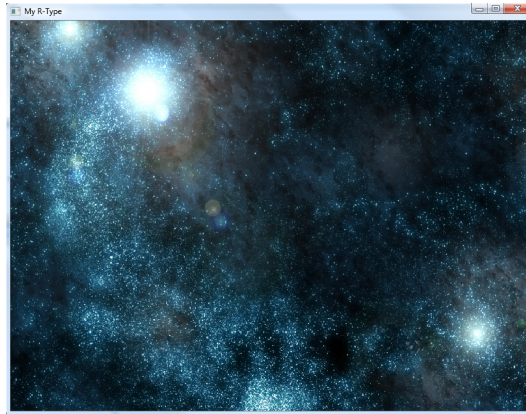
Avant toute chose il faut ouvrir une fenetre. Si vous utilisez un outil comme XNA cela devrait-être un jeu d'enfant enrevanche si vous utilisez DirectX ou OpenGL la tache devrait être bien plus hardue. Ne négligez pas cette étape un espace d'affichage mal configuré est souvent la cause de beaucoup de bugs et ou lenteur dans un jeux vidéo.

1.2 conseils

- **XNA** : Rien a faire tout est déjà près pour vous passez à l'étape suivante.
- **OpenTK** : créez une GameWindow n'utilisez pas de GLControl. (Les exemple d'openTK vous montrerons comment créer une fenêtre basique à partir de gameWindow).
- **SlimDX** : [Ouvrir une fenêtre avec slimSX](#)

2 Dessiner un arriere plan

2.1 objectifs



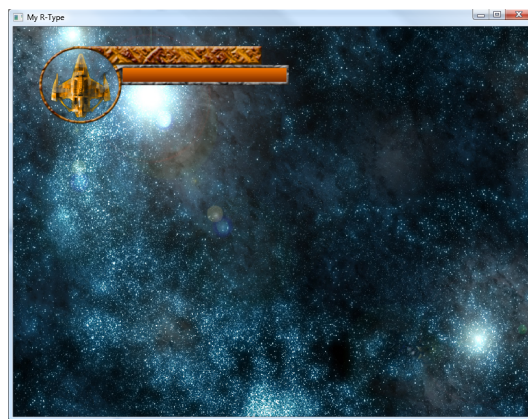
Dans cette étape il "suffit" d'afficher un sprite qui occupe l'intégralité de l'arrière plan. Profitez de cette étape pour vous familiariser avec les fonctions de chargement d'image d'image ainsi qu'avec les fonction de dessin. La gestion de l'ordre de dessin ou de la transparence n'est absolument pas nécessaire ici.

2.2 conseils

- **XNA** : Utilisez la méthode [SpriteBatch.Draw](#) pour dessiner un sprite et utilisez le [ContentManager](#) pour charger l'image à afficher
- **OpenGL et DirectX** : Dessiner un sprite avec OpenGL ou DirectX revient en fait à dessiner un polygone sur l'écran sans lui faire subir de transformation 3D et en lui appliquant une texture
- **OpenTK** : commencez par l'excellent [tutoriel de NeHe sur le dessin des polygones avec OpenGL](#) puis celui sur [le chargement de texture avec OpenTK](#).
- **SlimDX** : Lisez la [page sur le dessin de triangle avec slimDX](#) puis regardez du côté des exemples fournis avec le [SDK DirectX](#).
- **DirectX** : Regardez aussi du côté de D3DX avec [D3DXCreateSprite](#).

3 Dessiner des sprites

3.1 objectifs



C'est ici que les choses sérieuses commence ! Nous allons maintenant dessiner l'interface de notre jeux. Une fois de plus pour ceux qui ont choisit XNA cela devrait être aussi facile que de dessiner l'arrière plan. Pour ceux qui ont choisit DirectX ou OpenGL cela ce corse. Il va falloir mettre en place du blending. Il s'agit de la dernière étape 2D de ce workshop avant de passer à la 3D donc proffitez bien de cette étape pour finaliser toute votre interface afin que cela ne vous handicape pas plus tard.

3.2 conseils

- **XNA** : Rien de bien différent comparé à l'étape précédente, cette étape ne devrait pas poser de problème.

- **OpenGL et DirectX** : Il va vous falloir configurer le blending (mélange de couleur) afin de pouvoir gérer la transparence. Le blending doit être configuré de sorte que : $source = source * source.alpha$
 $destination = destination * (1.0 - source.alpha)$.
- **OpenGL** : utilisez [glBlendFunc](#) pour configurer le blending avec OpenGL.
- **DirectX** : quand à DirectX vous trouverez les fonctions pour configurer le Blending sur [MSDN](#).

4 Un peu de logique ...

4.1 objectifs

Pas de programmation graphique ici. Le but de cette étape est de donner vie à notre interface : implémentez la bare de vie, le passage de l'écran de démarrage à l'écran de jeu ...

4.2 conseils

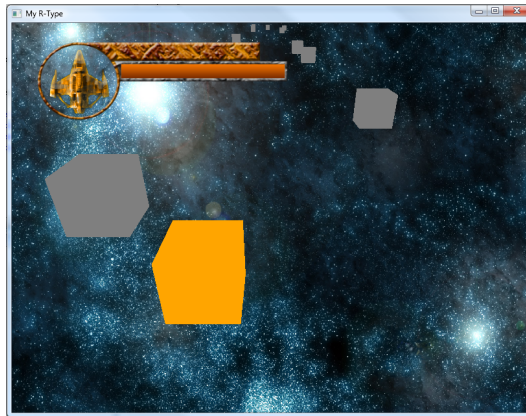
- mettez en application ce que vous avez appris pendant le workshop BDSP (si vous y étiez).

III Et maintenant de la 3D

En ce qui concerne la 3D et contrairement à la 2D nous ne recommandons pas l'utilisation de XNA. En effet ce dernier peut être plus frustrant qu'autre chose lorsqu'il s'agit de faire de la 3D. Cependant sachez que rien ne vous empêche d'utiliser XNA malgré tout.

1 Mon vaisseau c'est un cube !

1.1 objectifs



Cela peut sembler simple voire simpliste mais détrompez-vous, passer ce palier est probablement l'une des étapes les moins aisées du workshop. Pour pouvoir afficher ces cubes vous allez devoir :

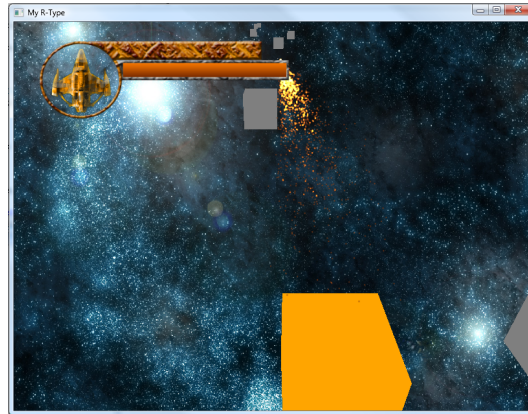
- Configurer et utiliser le Z-Buffer.
- Créer vos matrices de transformation ("lookAt" pour la caméra et "translate" pour les objets).
- Dessiner des polygones.

1.2 conseils

- **XNA** : Créez vos polygones en faisant un tableau de [VertexPosition-Color](#) et dessinez les avec [GraphicsDevice.DrawUserPrimitives](#).
- **OpenGL** : une fois de plus lisez [les articles de NeHe](#)
- **DirectX** : Les exemples du SDK DirectX contiennent toutes les informations pour comprendre comment réaliser cette tâche.

2 Des particules

2.1 objectifs



C'est bien tout ça mais ça manque un peu d'action. Nous allons maintenant faire en sorte que nos vaisseaux puissent tirer des projectiles. Pour cela nous allons utiliser des particules. Cette étape n'est pas vraiment indispensable pour la suite. Si vous la trouvez trop compliquée ou que vous voulez vite passer à la suite, utilisez simplement une sphère ou tout autre objet de votre choix afin de représenter les projectiles de vos vaisseaux et ceux des ennemis.

2.2 conseils

Cette étape est optionnelle. Utilisez la technique avec laquelle vous vous sentez le plus à l'aise. Les commandes ci-dessous ne sont que des suggestions. Cette étape n'est pas évidente à réaliser, ne bloquez pas dessus et passez à la suite si vous ne voyez pas comment faire.

- **XNA et DirectX** calculez la position à l'écran (en 2D donc) de l'endroit où vous souhaitez dessiner des particules (un point en 3D), puis dessinez vos particules sous forme de sprites.
- **OpenGL** dessinez des points avec `GL_POINTS` dans votre fonction de dessin en lieu et place de `GL_TRIANGLES`, puis faites les quelques ajustements nécessaires.

3 Charger de vrai objets

3.1 objectifs



C'est le moment de donner un coté un peu moins cubique a ce petit projet. Une fois de plus remplacer les cube par des objets sera une tache plus ou moins aisé selon vos choix. Et c'est sans nulle doute ceux qui ont choisit OpenGL qui auront le plus de pain sur la planche ici.

3.2 conseils

- **XNA** utilisez de nouveau le [ContentManager](#).
- **DirectX** utilisez [les outils D3DX](#).
- **OpenGL** il va vous falloir charger votre fichier "à la main". Utilisez format wavefront qui est très simple à lire. Bon courage.

4 Un peu de mouvement

4.1 objectifs

Cette étape ne contient pas vraiment de phase graphique. L'objectif est ici de donner vie à notre petit jeu :

- finalisez le déplacement de votre vaisseau et de celui de vos ennemis.
- finalisez l'apparitions des ennemis.
- gérez le score du joueur.
- rendez ce programme fun !

4.2 conseils

Faites vous plaisir !

IV Shaders

Cette dernière partie est beaucoup plus difficile que les précédentes et nous vous conseillons de vous y attaquer uniquement si vous vous sentez particulièrement à l'aise en programmation et plus spécifiquement en programmation graphique.

1 Lumière

1.1 objectifs



Ici nous allons enfin passé aux choses serieuses et nous allons utiliser des shaders. La première étape de cette sections consacré au shaders consiste à implémenté illumination de type Lambert (l'une des plus simples).

2 Ombrage celluloide

2.1 objectifs

"L'ombrage celluloide à été inventé par des informaticiens pour pallier l'absence de graphiste compétant dans certaines studio de jeux vidéo"¹.
Tout est dit !

2.2 conseils

- un ombrage celluloide est composé uniquement de quelques modification au modèle de Lambert.
- pour gérer les contours pensez à étudier la valeur du produit scalaire de la normale d'un point avec le vecteur donnant l'orientation de la caméra.

1. phrase de Pannos Baroudjan (Promo 2012) lors d'une de mes conversation avec lui, ma mémoire étant ce qu'elle est il se peut que les termes ne soient pas exactement ceux-ci mais l'idée est là

V Conclusion



Have fun !