Projet 3D

RUSH 2 – L'affichage du terrain

1. Buts : Afficher notre terrain correctement et la caméra 1ère personne pour le «parcourir»

2. Modalités: 2 équipiers sur l'affichage du terrain et deux (ou un) équipiers sur les caméras

3. Caméras : une caméra «free cam» et une caméra «level cam» (caméra de vue orthographique qui regarde le terrrain vue du ciel)

4. Trucs:
   - Pour le terrain, utiliser la fonction de chargement déjà écrite dans la partie 1 et écrire une classe CTerrain inspirée de CBlocEffet1.

TRUCS et INDICES pour la Partie 2

TERRAIN

0. Utiliser le programme du chapitre 6
1. Avoir une fonction de chargement qui fonctionne!
2. S'inspirer du constructeur de la classe **CBlocEffet1**
    - 2.0 Charger le terrain.
    - 2.1 Créer un «vertex buffer» de la bonne taille et du bon format
    - 2.2 Placer les sommets dans le vertex buffer
    - 2.3 Prendre en note le nombre de sommets (ou le calculer)
    - 2.4 Créer un «index buffer» de la bonne taille et du bon format
    - 2.5 Placer les index dans l'index buffer
    - 2.6 Prendre en note le nombre d'index (ou le calculer)
3. Modifier la fonction **Draw** pour qu'elle utilise le bon nombre d'entrées d'index.
4. Insérer un terrain dans la scène.
5. Modifier les transformations pour que l'affichage affiche votre terrain.
6. Attendre la caméra...

CAMÉRA

1. Faire le petit tutoriel!
2. Implanter le gestionnaire DirectInput (déjà fait avec chap 10)
3. Essayer la caméra dans le prog. du chapitre 6
4. Adapter la caméra avec le terrain.

TRUCS et INDICES pour la Partie 2

Une classe de caméra

Attention - Main droite

Ce sont des indices, le code présenté peut demander des ajustements

Construire une classe CCamera qui implantera une caméra 1ère personne (à peu près...).

Fonctions importantes:

```
CCamera ( position, direction , up, ... )
SetPosition
SetDirection
Update
```
-> Met à jour la matrice de vision du système

0. Déclarons la classe de caméra

```
class CCamera
{
    …
};
```

1. Nous allons commencer avec `Update()`
   Nous voulons y placer un code similaire à celui que nous faisions dans CMoteur::InitScene:

```
XMMATRIX matView;

// Matrice de la vision
matView = XMMatrixLookAtRH(   XMVectorSet( 0.0f, 3.0f,-5.0f, 1.0f ),
                              XMVectorSet( 0.0f, 0.0f, 0.0f, 1.0f ),
                              XMVectorSet( 0.0f, 1.0f, 0.0f, 1.0f ) );

// Recalculer matViewProj
matViewProj = matView * matProj;
```

   Mais nous le modifierons pour tenir compte de la position, de la direction et du vecteur up:

```
// Matrice de la vision
*pMatView = XMMatrixLookAtRH( position,
                              (position + direction),
                              up );

// Recalculer matViewProj
*pMatViewProj = (*pMatView) * (*pmatProj);
```

   La variable pMoteur ... Nous la déclarons dans la classe de caméra avec la position, la direction et le up et nous l'initialiserons dans le constructeur paramétré ou ailleurs... (*voir plus loin*).

```
XMVECTOR position;
XMVECTOR direction;
XMVECTOR up;
XMMATRIX* pMatView;
XMMATRIX* pMatProj;
XMMATRIX* pMatViewProj;
```

2. Le constructeur paramétré:
   On règle ça comme ça:

```
CCamera::CCamera(const XMVECTOR& position_in,
                 const XMVECTOR& direction_in,
                 const XMVECTOR& up_in,
                 XMMATRIX* pMatView_in,
                 XMMATRIX* pMatProj_in,
                 XMMATRIX* pMatViewProj_in)
{
    Init(position_in, direction_in, up_in, pMatView_in, pMatProj_in, pMatViewProj_in);
}
```

   Mais on implémente la fonction **Init**:

```
void CCamera::Init(const XMVECTOR& position_in,
                   const XMVECTOR& direction_in,
                   const XMVECTOR& up_in,
                   XMMATRIX* pMatView_in,
                   XMMATRIX* pMatProj_in,
                   XMMATRIX* pMatViewProj_in)

{
   pMatView = pMatView_in;
   pMatProj = pMatProj_in;
   pMatViewProj = pMatViewProj_in;

   position = position_in;
   direction = direction_in;
   up = up_in;
}
```

3. Les autres fonctions (facile):

```
void SetPosition(const XMVECTOR& position_in){position = position_in;};
void SetDirection(const XMVECTOR& direction_in){direction = direction_in;}
void SetUp(const XMVECTOR& up_in){up = up_in;}
```

4. On teste si ça marche:

   a) On enlève les lignes de InitScene (ou on les met en commentaire)

```
//      XMMATRIX matView;
//
//      // Matrice de la vision
//      matView = XMMatrixLookAtLH(   XMVectorSet( 0.0f, 3.0f,-5.0f, 1.0f ),
//                                    XMVectorSet( 0.0f, 0.0f, 0.0f, 1.0f ),
//                                    XMVectorSet( 0.0f, 1.0f, 0.0f, 1.0f ) );
//
//      // Recalculer matViewProj
//      matViewProj = matView * matProj;
```

   b) On peut maintenant faire l'initialisation de la caméra dans CMoteur::InitTransformations

```
camera.Init( XMVectorSet(0.0f, -10.0f, -10.0f, 1.0f),
             XMVectorSet (0.0f, 1.0f, 0.0f, 1.0f),
             XMVectorSet (0.0f, 0.0f, 1.0f, 1.0f),
             this);

camera.Update();
```

   avant ou après le code déjà en place... N'oubliez pas de déclarer la caméra dans la classe de moteur!!!

```
CCamera camera;
```

5. Tourner la caméra : on utilise les touches de clavier ← et → :

   a) On implémente **CDIManipulateur** et on l'ajoute à notre programme… voir chapitre 10...

   b) On ajoute la variable **angleDirectionCamera** dans CMoteur (ou dans la classe de caméra…):

```
float angleDirectionCamera;
```

   c) On l'initialise dans **CMoteur::InitScene** :

```
angleDirectionCamera = XM_PI/2.0f;
```