Initiation DirectX

Le but de ce TD est de vous familiariser avec les fonctions DirectX. Pour cela vous créerez une fenêtre simple puis vous afficherez des sprites à l'intérieur de cette fenêtre.

Instructions :

1°) Le fichier CreateDevice.cpp est fournis dans le dossier de tutoriels officiels de DirectX (tutoriels que vous pouvez trouver à partir du dossier DirectX dans "Samples\C++\Direct3D\Tutorials") ; il permet de créer une fenêtre en DirectX.

A partir de ce fichier créez une classe Game qui permettra de créer une fenêtre.

2°) Compléter la classe Sprite avec les fonctions suivantes :

- SetPosition
- SetTexture
- SetRectangle
- SetCenter
- SetRotation
- SetScaleCenter
- SetScale
- SetColor

Afin d'afficher un sprite dans la fenêtre.

Questions subsidiaires :

3°) Ajoutez une fonction Move à la classe Sprite afin de bouger votre sprite dans la fenêtre (indice : utilisez ::GetAsyncKeyState)

4°) Complétez la classe SpriteManager(fournie dans l'énoncé) qui permettra de gérer plusieurs sprites. Utilisez ce manager pour afficher 5 sprites à l'écran (un dans chaque coin et un au centre de l'écran).

Informations :

Afin de ne pas inclure les dossiers "include" et "lib" dans chacun de vos projets, configurez Visual C++ de manière à ce qu'il les considère comme des dossier dans lesquels aller chercher des informations à chaque génération de projet.
Pour cela allez dans "Outils → Options → Projets et solutions → Répertoires de VC++" et ajoutez les dossiers de DirectX nécessaires (Program Files → Microsoft DirectX SDK).
De plus ajoutez cette ligne : "d3dxof.lib dxguid.lib d3dx9.lib d3d9.lib winmm.lib dxerr.lib dxguid.lib dxerr.lib" dans les dépendances supplémentaires de votre projet. Ce sont les lib dont vous aurez besoin dans chacun de vos projets DirectX.

Lors de la génération d'un projet DirectX, vous pouvez rencontrer une erreur de type "Error loading d3dx9_42.dll". Cela veut dire que la version de DirectX que vous utilisez est différente de la version que vous avez utilisée la dernière fois que vous avez généré le projet. Le problème est généralement réglé en nettoyant la solution (Générer → Nettoyer la solution).

A propos de la fonction MsgProc : cette fonction récupère les messages envoyés à l'application (lors de l'agrandissement de la fenêtre par exemple).
Lors de la création de la fenêtre dans la fonction wWinMain, le troisième paramètre de la structure WNDCLASSEX est un pointeur sur la fonction MsgProc.

Documentation :

D3DXVECTOR2 Structure

Describes a two-component vector including operator overloads and type casts.

Syntax :

```
typedef struct D3DXVECTOR2 {
  FLOAT x;
  FLOAT y;
} D3DXVECTOR2, *LPD3DXVECTOR2;
```

Parameters :

x
FLOAT
The x-component

y
FLOAT
The y-component

D3DXVECTOR3 Structure

Describes a three-component vector including operator overloads and type casts.

Syntax :

```
typedef struct D3DXVECTOR3 {
  FLOAT x;
  FLOAT y;
  FLOAT z;
} D3DXVECTOR3, *LPD3DXVECTOR3;
```

Parameters :

x
FLOAT
The x-component.

y
FLOAT
The y-component.

z
FLOAT
The z-component.

D3DCOLOR

Defines the fundamental Direct3D color type.

Syntax :

```
typedef DWORD D3DCOLOR;
```

A four byte value that typically represents the alpha, red, green, and blue components of a color. It can also represent luminance and brightness.

You can set the D3DCOLOR type using one of the following macros.

- D3DCOLOR_ARGB
- D3DCOLOR_AYUV
- D3DCOLOR_COLORVALUE
- D3DCOLOR_RGBA
- D3DCOLOR_XRGB
- D3DCOLOR_XYUV

D3DXCreateTextureFromFile Function

Creates a texture from a file.

Syntax :

```
 HRESULT  D3DXCreateTextureFromFile(
  __in   LPDIRECT3DDEVICE9 pDevice,
  __in   LPCTSTR pSrcFile,
  __out  LPDIRECT3DTEXTURE9 *ppTexture
);
```

Parameters :

pDevice [in]
: LPDIRECT3DDEVICE9
Pointer to an IDirect3DDevice9 interface, representing the device to be associated with the texture.

pSrcFile [in]
: LPCTSTR
Pointer to a string that specifies the filename. If the compiler settings require Unicode, the data type LPCTSTR resolves to LPCWSTR. Otherwise, the string data type resolves to LPCSTR. See Remarks.

ppTexture [out]
: LPDIRECT3DTEXTURE9
Address of a pointer to an IDirect3DTexture9 interface, representing the created texture object.

Return Value :

HRESULT

If the function succeeds, the return value is D3D_OK. If the function fails, the return value can be one of the following: D3DERR_NOTAVAILABLE, D3DERR_OUTOFVIDEOMEMORY, D3DERR_INVALIDCALL, D3DXERR_INVALIDDATA, E_OUTOFMEMORY.

D3DXCreateSprite Function

Creates a sprite object which is associated with a particular device. Sprite objects are used to draw 2D images to the screen.

Syntax :

```
 HRESULT  D3DXCreateSprite(
  __in   LPDIRECT3DDEVICE9 pDevice,
  __out  LPD3DXSPRITE *ppSprite
);
```

Parameters :

pDevice [in]
: LPDIRECT3DDEVICE9
Pointer to an IDirect3DDevice9 interface, the device to be associated with the sprite.

ppSprite [out]
: LPD3DXSPRITE
Address of a pointer to an ID3DXSprite interface. This interface allows the user to access sprite functions.

Return Value :

HRESULT

If the function succeeds, the return value is S_OK. If the function fails, the return value can be one of the following: D3DERR_INVALIDCALL, E_OUTOFMEMORY.

Rect Structure

Describes the width, height, and location of a rectangle.

Parameters :

x
: The x-coordinate location of the left side of the rectangle.

y
: The y-coordinate location of the top side of the rectangle.

width
: A non-negative value that represents the Width of the rectangle.

height
: A non-negative value that represents the Height of the rectangle.

GetAsyncKeyState Function

Determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to **GetAsyncKeyState**.

Syntax :

```
SHORT WINAPI GetAsyncKeyState(
  __in  int vKey
);
```

Parameters

vKey [in]
int

The virtual-key code. For more information, see Virtual Key Codes.

You can use left- and right-distinguishing constants to specify certain keys. See the Remarks section for further information.

Return Value

SHORT

If the function succeeds, the return value specifies whether the key was pressed since the last call to **GetAsyncKeyState**, and whether the key is currently up or down. If the most significant bit is set, the key is down, and if the least significant bit is set, the key was pressed after the previous call to **GetAsyncKeyState**. However, you should not rely on this last behavior; for more information, see the Remarks.

The return value is zero for the following cases:

- The current desktop is not the active desktop
- The foreground thread belongs to another process and the desktop does not allow the hook or the journal record.

Source : MSDN (http://msdn.microsoft.com/en-us/library)