Introduction à l’API Win32 et DirectX

Sebti Foufou

Contenu du cours

I. Le minimum nécessaire pour comprendre la programmation avec l'API Win32 de Microsoft
II. Les outils GDI de l'API Win32
III. Généralités sur DirectX
IV. Adapter une application API Win32 pour utiliser DirectX
V. Manipuler des maillages avec Direct 3D
VI. Bases de la visualisation en 3D

I. API Win32 de Microsoft

- Application console

```
#include <stdio.h>

int main(void) {
      printf("Hello world!¥n");
      return 0;
}
```

- Application graphique mais console

```
#include <windows.h>
int main(void) {
   MessageBox(NULL, "Hello world!", "Sample", MB_OK);
   return 0;
}
```

- Application Win32

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
   MessageBox(NULL, "Hello world!", "Sample", MB_OK);
   return 0;
}
```

Prog. API Win32

- La fonction WinMain: elle remplace le main

```
int WINAPI WinMain(HINSTANCE hInst,
                   HINSTANCE hPrevInst,
                   LPSTR lpCmdLine,
                   int nCmdShow);
```

- hInts: handlede l'instance de l'application (entier permettant d'identifier l'application de manière unique.).
- hPrevInst [obsolète] valeur toujours nulle (WIN16 : handlesur l'instance précédente)
- lpCmdLine: chaîne de caractères contenant les paramètres passés à la ligne de commande (exemple: "/fullscreen").
- CmdShow: flag indiquant comment la fenêtre sera affichée à sa création(un raccourci peut par exemple préciser que l'application doit être lancée iconifiée).

Ce qu’on fait dans le WinMain

1. Initialiser l'application.

   à savoir les propriétés de la fenêtre et le nom de la fonction de gestion des messages.
2. Afficher sa fenêtre principale.
3. Entrer la boucle de traitement des messages. La sortie de cette boucle a lieu lorsque le message WM_QUIT est reçu. (éventuellement envoyé par l'application elle-même).
4. A la sortie, libérer les ressources allouées à l'application.

Exemple

```
#include <windows.h>    //basic windows header file

// the entry point for any Windows program
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nShowCmd)
{
    //"Hello World" message box using MessageBox()
    MessageBox(NULL,
               L"Hello World!",
               L"Just another Hello World program!",
               MB_ICONEXCLAMATION | MB_OK);

    // return 0 to Windows
    return 0;
}
```

MessageBox()

```
int MessageBox(HWND hWnd,
               LPCTSTR lptext,
               LPCTSTR lpcaption,
               UINT utype);
```

Cette fonction ouvre une boite de dialogue.

HWND hWnd : le handle d'une fenêtre ou NULL

LPCTSTR lptext : pointeur vers 16 bits contenant le texte du message à afficher dans la boite

LPCTSTR lpcaption : pointeur vers 16 bits contenant le titre de la fenêtre

UINT utype : détermine le style de la boite de dialogue. Ce style peut être une combinaison de ces valeurs : MB_CANCELTRYCONTINUE, MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO, MB_YESNOCANCEL. Pour afficher une icône dans la boîte de dialogue, spécifier l'une de ces valeurs : MB_ICONCONFIRATION, MB_ICONEXCLAMATION, MB_ICONERROR.

Valeurs retournées : IDCANCEL, IDTRYAGAIN, IDCONTINUE, IDNO, IDYES, IDOK

Construire une application Win32

1. Création d'une classe fenêtre à partir de laquelle seront dérivées nos fenêtre.
2. Enregistrement de la classe fenêtre.
3. Création de la fenêtre construite à partir des attributs de la classe fenêtre dont elle dérive et de ses propriétés individuelles.
4. Ecriture d'une fonction de messages événementiels qui sera appelée lorsque le système ou l'utilisateur interagit avec notre fenêtre. Le nom de la fonction est donné lors de la création de la fenêtre.
5. Affichage à l'écran de la fenêtre
6. Création d'une boucle de messages qui appellera notre fonction de messages évènementiels (mécanisme de callBack).

Construire la fenêtre

- Par l’intermédiaire de ces trois fonctions:
  - RegisterClassEx();
  - CreateWindowEx();
  - ShowWindow();

1. La classe Fenêtre

```
Typedef struct tagWNDCLASSEX
{
    UINT cbSize;
    UINT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HANDLE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrbackGround;
    LPCTR lpszMenuName;
    LPCTR lpszClassName;
    HICON hIconSm;
} WNDCLASSEX;
```

Exemple

```
WNDCLASSEX myWndClass = {
    sizeof(WNDCLASSEX),
    CS_HREDRAW | CS_VREDRAW,
    WndProc,
    0, 0,
    hInstance,
    NULL,    // ou LoadIcon(NULL, IDI_APPLICATION)
    NULL,    // ou LoadCursor(NULL, IDC_ARROW)
    NULL, NULL,
    "MaClasseFenetre",
    NULL    // ou LoadIcon(NULL, IDI_APPLICATION)
};
```

On peut aussi renseigner les champs individuellement:

```
/ ...
myWndClass.cbSize = sizeof(WNDCLASSEX);
myWndClass.style = CS_HREDRAW | CS_VREDRAW;
// ..
```

2. Enregistrer la classe de notre fenêtre

```
// this struct holds information for the window class
WNDCLASSEX wc;

// clear out the window class for use
ZeroMemory(&wc, sizeof(WNDCLASSEX));

// fill in the struct with the needed information
wc.cbSize = sizeof(WNDCLASSEX);
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc = (WNDPROC)WindowProc;
wc.hInstance = hInstance;
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
wc.lpszClassName = L"WindowClass1";

// register the window class
RegisterClassEx(&wc);
```

3. Créer la fenêtre

On a souvent besoin d'une seule fenêtre, on utilisera la fonction CreateWindowEx pour la créer.

```
WND CreateWindowEx(DWORD dwExStyle,
                   LPCTSTR lpClassName,
                   LPCTSTR lpWindowName,
                   DWORD dwStyle,
                   int x,
                   int y,
                   int nWidth,
                   int nHeight,
                   HWND hWndParent,
                   HMENU hMenu,
                   HINSTANCE hInstance,
                   LPVOID lpParam
);
```

Exemple

```
// create the window and use the result as
// the handle
hWnd = CreateWindowEx(NULL,
   L"WindowClass1",    // name of the window class
   L"Our First Windowed Program", //window title
   WS_OVERLAPPEDWINDOW,    // window style
   300,    // x-position of the window
   300,    // y-position of the window
   500,    // width of the window
   400,    // height of the window
   NULL,   // we have no parent window, NULL
   NULL,   // we aren't using menus, NULL
   hInstance,    // application handle
   NULL);    // used with multiple windows, NULL
```

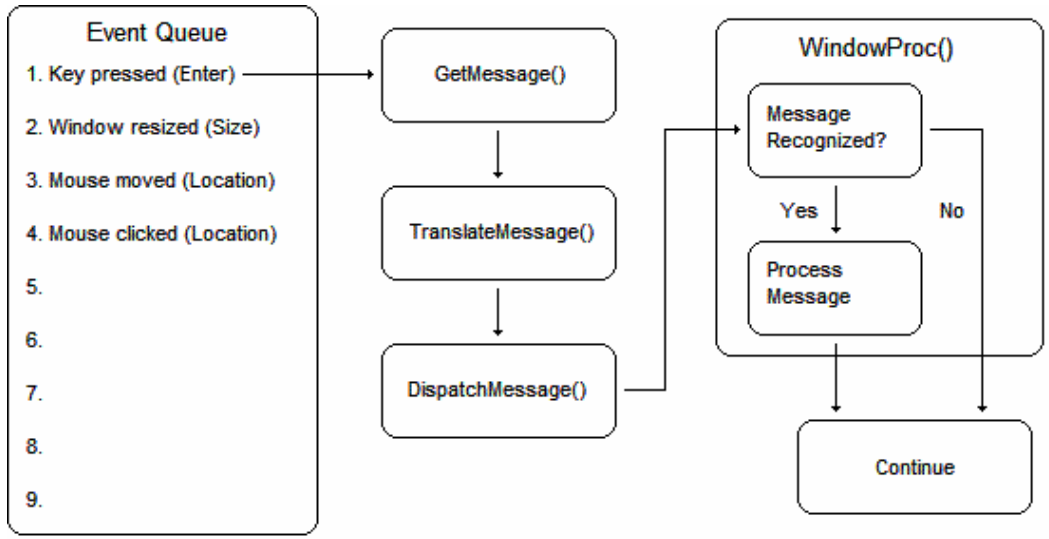
Lancer la fenêtre

La fenêtre est créée il ne reste plus qu'à l'afficher avec la fonction ShowWindow

```
BOOL ShowWindow(HWND hWnd,int nCmdShow);
```

Evènements et Messages

- La gestions des événements se fait en deux parties :
  - La boucle principale
  - La fonction WindowProc()



La boucle principale

```
// this struct holds Windows event messages
MSG msg;

// wait for the next message in the queue,
// store the result in 'msg'
while(GetMessage(&msg, NULL, 0, 0))
{
    // translate keystroke messages into the
    // right format
    TranslateMessage(&msg);

    // send the message to the WindowProc function
    DispatchMessage(&msg);
}
```

La fonction WindowProc()

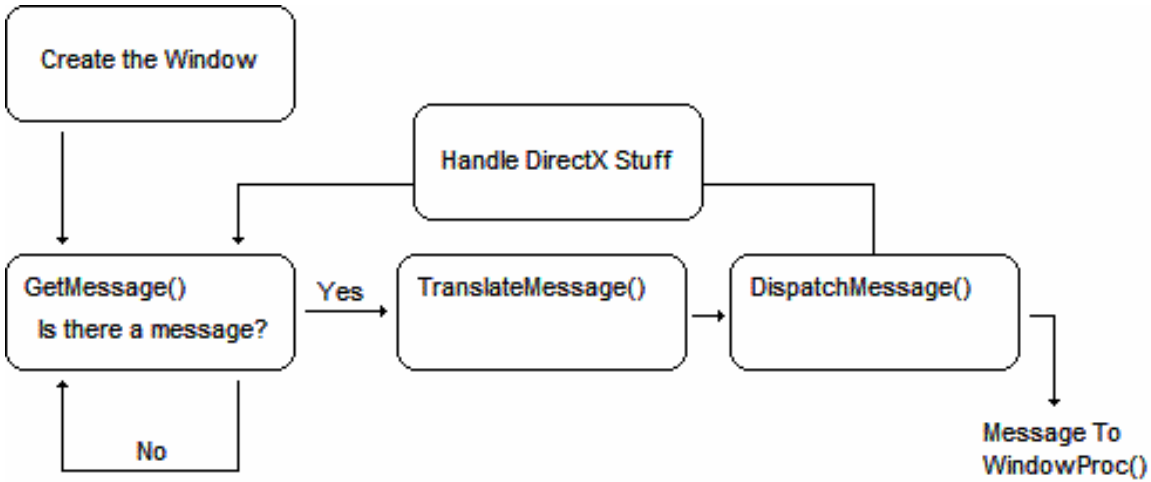
Entête:

```
LRESULT CALLBACK WindowProc(HWND hWnd,
                            UINT message,
                            WPARAM wParam,
                            LPARAM lParam);
```

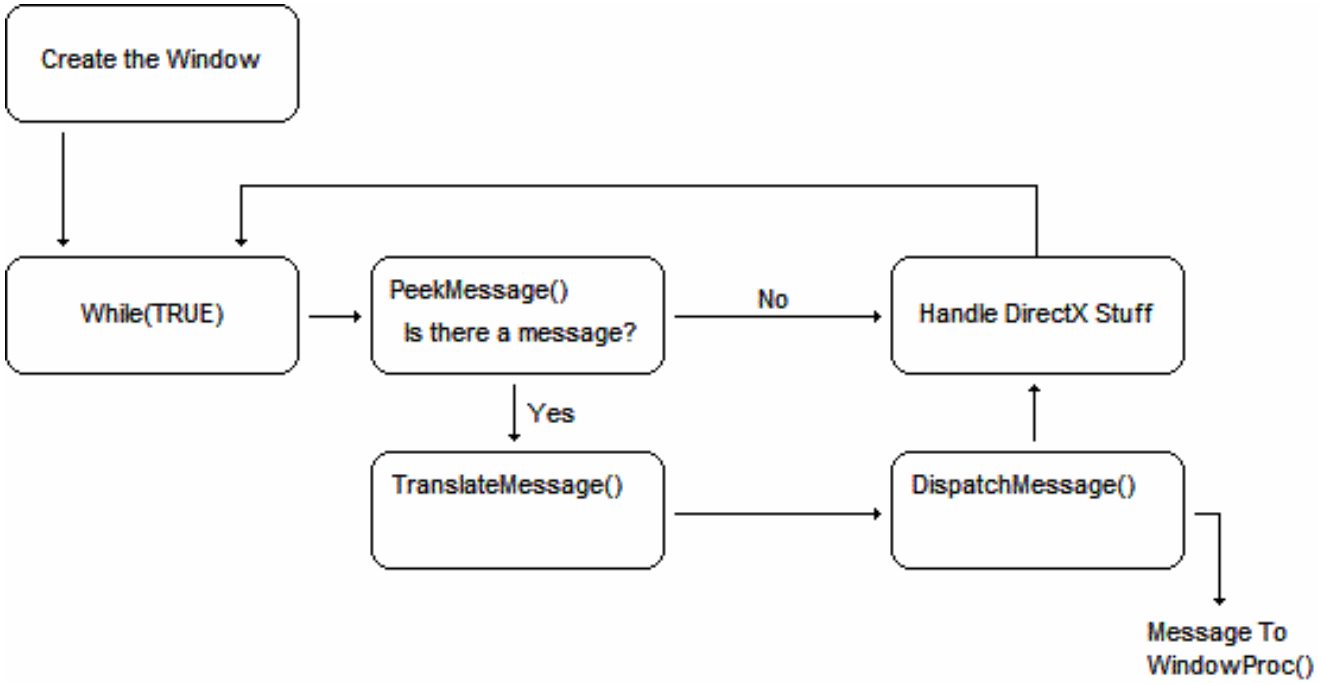
Code : un switch avec un paquet de case

```
// sort through and find what code to run for the message given
    switch(message)
    {
        // this message is read when the window is closed
        case WM_DESTROY:
        { // close the application entirely
          PostQuitMessage(0); return 0;
        } break;
        case … :
        …
    }
```

PeekMessage() au lieu de GetMessage()



PeekMessage() au lieu de GetMessage()



PeekMessage() au lieu de GetMessage()

Entête de la fonction PeekMessage

```
BOOL PeekMessage(LPMSG lpMsg,
                 HWND hWnd,
                 UINT wMsgFilterMin,
                 UINT wMsgFilterMax,
                 UINT wRemoveMsg
);
```

PeekMessage() au lieu de GetMessage()

Exemple de boucle avec la fonction PeekMessage

```
// Enter the infinite message loop
while(TRUE) {
    // Check to see if any messages are waiting in the queue
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
        // If the message is WM_QUIT, exit the while loop
        if (msg.message == WM_QUIT)
            break;

        // Translate the message and dispatch it to WindowProc()
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    // Run game code here
    // ...
    // ...
}
```

Remarques

- Pour obtenir la taille de la fenêtre de visualisation :

  RECT clientRect;

  GetClientRect(hWnd,&clientRect);

- Pour forcer le réaffichage
  - Envoyer le message WM_PAINT par appel à la fonction InvalidateRect
  - InvalidateRect(hWnd,NULL,TRUE);

II. L’interface graphique GDI de l’API Win32

Plan

- Affichage de texte
- Affichage des pixels
- Tracer des lignes
- Tracer des formes pleines
- Les plumes et les brosses

Le texte

BOOL TextOut(HDC hdc, int x, int y, LPCSTR lpString, int cbString);

- Hdc : un handle du périphérique. Utiliser BeginPaint pour obtenir le handle, et EndPaint pour le libérer. On peut aussi le déclarer en variable globale et utiliser GetDC(hWnd), hWnd est le handle de la fenetre principale
- lpString: le texte à afficher

Exemple : pour afficher « Hello World » à la position 10,10 de la fenêtre de visualisation, on fait :

```
PAINTSTRUCT ps;
HDC hdc = BeginPaint(hWnd, &ps);
TextOut(hdc,10,10,"Hello World",11);
EndPaint(hWnd, &ps);
```

Manipuler des pixels directement

- On peut afficher un pixel avec la fonction SetPixel:

  SetPixel(HDC hdc, int X, int Y, COLORREF crColor);

- Hdc : handle du périphérique
- X, Y: les coordonnées du pixel
- COLORREF : le type win32 de représentation des couleurs. On utilisera la macro RGB pour créer des instances de COLORREF à partir des composant Rouge, Vert, Bleu.

- Exemple : afficher un pixel rouge à la position 100, 100, on fait :

  SetPixel(hdc, 100, 100, RGB(255,0,0));

Tracer des lignes

- Plusieurs possibilités:
  - MoveTo, LineTo
  - PolyLine,

  Polyline(HDC hdc, CONST POINT *lppt, int cPoints);
  - PolyPolyLine, Arc, …

- Exemple :

```
POINT pntArray[2];
pntArray[0].x=10;
pntArray[0].y=10;
pntArray[1].x=100;
pntArray[1].y=100;
Polyline(hdc, pntArray, 2);
```

Tracer des formes pleines

- FillRect : rectangle plein (e.g. intérieur coloré en vert)

  FillRect(HDC hDC, CONST RECT *lprc, HBRUSH hbr);

- Ellipses : ellipse plein

  BOOL Ellipse(HDC hdc, int nLeftRect, int nTopRect, int nRightRect, int nBottomRect);

- hbr = la brosse utilisée pour le remplissage. La dispositive suivante montre comment créer et utiliser les plumes et les brosses.

Les plumes

- Création:
  - HPEN CreatePen(int fnPenStyle, int nWidth, COLORREF crColor)
  - CreatePenIndirect : pour créer une plume à partir d'une plume constantes déjà définie dans l'API
- Les styles possibles : PS_SOLID PS_DASH PS_DOT PS_DASHDOT PS_DASHDOTDOT PS_NULL (plumes invisible).

- Exemple : une plume solide d'un pixel d'épaisseur :

  HPEN greenPen=CreatePen(PS_SOLID, 1, RGB(0,255,0));

- La fonction SelectObject permet de sélectionner une plume en vue de son utilisation pour dessiner. Elle retourne un pointeur sur l'ancienne plume (qu'il faut sauvegarder).

- Exemple :

  // Select our green pen into the device context and remember previous pen
  HGDIOBJ oldPen=SelectObject(hdc,greenPen);

  Polyline(hdc, pntArray, 2); // Draw our line

  // Select the old pen back into the device context
  SelectObject(hdc,oldPen);

Les brosses

- Création :
  - CreateSolidBrush : une brosse avec une couleur continue.

    HBRUSH blueBrush=CreateSolidBrush(RGB(0,255,0));
  - CreateBrushIndirect : une brosse avec un style prédéfini.
  - CreatePatternBrush : une brosse avec une bitmap.

- Exemple remplir un rectangle avec la blueBrush :

```
RECT rct;
rct.left=10;   rct.right=100;
rct.top=10;   rct.bottom=200;
FillRect(hdc, &rct, blueBrush);
```

III. Généralités sur DirectX

DirectX

- Une bibliothèque de développement multimédia.
- Constructeur : Microsoft
- Gratuit
- DirectX permet d'assurer une compatibilité logicielle entre diverses configurations matérielles.
- DirectX fournit une interface logicielle standard, destinée à surmonter ces obstacles matérielles dans la plupart des cas.
- Tous les principaux fabricants de matériel proposent des pilotes de périphériques pour DirectX.

Composants de DirectX

- Le kit de développement DirectX regroupe une grande variété de composants.
- En fonction de l'application on peut sélectionner les fonctionnalités dont on a besoin.
- Ce kit comprend suffisamment d'interfaces pour répondre à tous vos besoins pour la création d'oeuvres multimédias et de jeux.
- Les outils du kit se répartissent en deux lots :
  - Le kit de développement SDK DirectX
  - Le kit de développement SDK DirectX Media.

Le kit de développement SDK DirectX

- DirectDraw: pour le dessin 2D

  DirectDraw permet d'exploiter les capacités 2D d'une carte graphique tout en autorisant l'utilisant des services du GDI (Graphic Device Interface), l'API graphique de Windows. Son point fort est de permettre la manipulation de la mémoire vidéo sous la forme de surfaces de résolutions diverses, une approche efficace qui se retrouve entre autres sur la console Playstation.

- DirectSound : pour le son

  DirectSound permet d'exploiter les capacités d'entrée-sortie d'une carte son pour numériser, mixer des échantillons sonores et en restituer le résultat, éventuellement agrémenté d'effets 3D perfectionnés.

- DirectInput : pour gérer les entrées utilisateur

  DirectInput permet de récupérer les informations transmises par l'utilisateur, que ce soit par l'intermédiaire de la souris, du clavier ou du joystick. Elle permet aussi de renvoyer des données vers des périphériques à retour de force.

Le kit de développement SDK DirectX

- Direct3D : pour le dessin 3D

  Direct3D permet de créer des scènes de 3D, d'en manipuler les objets avec une palette de services et de les afficher en utilisant les capacités 3D d'une carte graphique quand elle en dispose. Direc3D dispose de deux modes de d'utilisation :
  - Le mode immédiat permet de gérer la transformation, l'illumination et le rendu de simples primitives (triangles, points, droites).
  - Le mode retenu met à la disposition du développeur un moteur 3D complet capable de gérer des objets.
- DirecPlay : dialogue réseau

  DirectPlay permet de mettre en place et d'administrer un dialogue point à point ou client-serveur entre plusieurs utilisateurs désirant évoluer simultanément dans un même contexte.
  - Approche centralisée : le serveur assure la gestion du contexte. Les clients modifient le contexte en soumettant des requêtes.
  - Approche décentralisée : chacun des utilisateurs peut modifier le contexte en local mais droit ensuite le transmettre aux autres utilisateurs.

Le kit de développement SDK DirectX

- DirectSetup : pour l'installation

  DirectSetup permet de commander l'installation de la partie distribuée de DirectX depuis une application.

- DirectMusic : pour la musique

  DirectMusic permet de rejouer des morceaux de musique faisant appel à des instruments synthétisés en temps réel à partir d'échantillons disponibles par défaut ou téléchargés dans la carte. DirectMusic supporte le format MIDI, il permet de composer à la volée des morceaux de musique.

Le kit de dév. SDK DirectX Media

- DirectX Transform : transformation en 2D et en 3D

  DirectX Transform pour la création d'effets graphiques dynamiques en 2D et en 3D, notamment pour la fusion des couches alpha et les distorsions de surface. Au cœur Chromeffects, DirectX Transform est destiné à la création de contenus Web, de logiciels de divertissement et mêmes de filtres pour Photoshop.

- DirectAnimation : pour les animations

  DirectAnimation est un ensemble de contrôles (chemins de déplacement, sprites, éléments de 3D, séquenceurs) qu'il est possible de référencer depuis une page Web. Ces contrôles disposent d'API qui permettent de les manipuler via un script JScript ou VBscript ou encore une applet Java. Il est possible d'intégrer ces contrôles dans une application développée en Java, en Visual Basic ou en C++.

- DirectShow : pour jouer les séquences vidéo

  DirectShow permet de jouer des séquences vidéo et audio (accéder en local ou à distance) et d'en capturer via des périphériques spécialistes. DirectShow supporte de nombreux formats : MPEG-1, WAV, MIDI, MPEG-2, PCM, DSS, AVI, ...

Utiliser DirectX

- Pour permettre la compilation en DirectX, vous devez :
  - Configurer l'ordinateur pour trouver les fichiers DirectX.
  - Lier les bibliothèques dans l'application selon les paramètres du projet (opération nécessaire pour chaque application construite avec DirectX)
  - Inclure les fichiers d'en-tête pour les bibliothèques DirectX (opération nécessaire pour chaque fichier source accédant aux composants DirectX).

- La plupart des applications DirectX sont écrites en C++ avec l’API Win32.

IV. Adapter votre application Win32 pour utiliser Direct3D

Premier programme Direct3D

- Définir les variables globales et les prototypes des fonctions utiles
- Définir une fonction pour initialiser Direct3D et créer un périphérique Direct3D
- Définir une fonction de rendu
- Définir la fonction de fermeture de Direct3D

Variables globales!! Pourquoi?

- L'objet Direct3D et le pointeur sur le périphérique seront utilisés dans tout le code, donc on les déclare globales

```
LPDIRECT3D9 d3dObject=NULL;
LPDIRECT3DDEVICE9 d3dDevice=NULL;
```

- LPDIRECT3D9 est un pointeur défini comme un typedef de IDirect3D9*
- Idem, LPDIRECT3DDEVICE9 est un pointeur défini comme un typedef de IDirect3DDevice9*
- Pour créer l'objet Direct3D:

```
d3dObject=Direct3DCreate9(D3D_SDK_VERSION);
```

- Pour créer le périphérique, il faut appeler la fonction

```
HRESULT IDirect3D9::CreateDevice(UINT adapter,
D3DDEVTYPE deviceType, HWND focusWindow,
DWORD behaviourFlags,
D3DPRESENT_PARAMETERS *presentationParameters,
IDirect3DDevice9** device);
```

1. Variables et fonctions globales

```
// include the basic windows header files and the Direct3D header file
#include <windows.h>
#include <windowsx.h>
#include <d3d9.h>

// include the Direct3D Library file
#pragma comment (lib, "d3d9.lib")

// global declarations
LPDIRECT3D9 d3d;    // the pointer to our Direct3D interface
LPDIRECT3DDEVICE9 d3ddev;    // the pointer to the device class

// function prototypes
void initD3D(HWND hWnd);    // sets up and initializes Direct3D
void render_frame(void);    // renders a single frame
void cleanD3D(void);    // closes Direct3D and releases memory

// the WindowProc function prototype
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam,
    LPARAM lParam);
```

2. Initialiser Direct3D

- 2 étapes :
  - Créer un objet Direct3D
  - Créer un périphérique pour Direct3D

- On va faire ce travail dans une fonction InitD3D.
- La fonction de création de périphérique (CreateDevice) nécessite un grand nombre de paramètres. Notamment un paramètre de type D3DPRESENT_PARAMETERS qui contient les paramètres de présentation de la scènes.
- Exemple: Initialiser D3DPRESENT_PARAMETERS

Exemple : D3DPRESENT_PARAMETERS

```
D3DPRESENT_PARAMETERS presParams;
ZeroMemory(&presParams,sizeof(presParams));


presParams.Windowed=TRUE;
presParams.SwapEffect=D3DSWAPEFFECT_DISCARD;
presParams.BackBufferFormat=D3DFMT_UNKNOWN;
presParams.PresentationInterval=D3DPRESENT_INTERVAL_ONE;
```

La fonction CreateDevice

```
HRESULT CreateDevice(
    UINT Adapter,
    D3DDEVTYPE DeviceType,
    HWND hFocusWindow,
    DWORD BehaviorFlags,
    D3DPRESENT_PARAMETERS *pPresentationParameters,
    IDirect3DDevice9 **ppReturnedDeviceInterface
);
```

- adapter: par défaut : D3DADAPTER_DEFAULT
- deviceType: hardware (D3DDEVTYPE_HAL) ou software (D3DDEVTYPE_SW).
- hFocusWindow : le handle de la fenêtre associée au périph.
- behaviourFlags : D3DCREATE_HARDWARE_VERTEXPROCESSING - D3DCREATE_SOFTWARE_VERTEXPROCESSING - D3DCREATE_MIXED_VERTEXPROCESSING

2. Initialiser Direct3D

```
// this function initializes and prepares Direct3D for use
void initD3D(HWND hWnd) {
    d3d = Direct3DCreate9(D3D_SDK_VERSION);    // create the Direct3D interface

    D3DPRESENT_PARAMETERS d3dpp;    // create a struct to hold device data

    ZeroMemory(&d3dpp, sizeof(d3dpp));    // clear out the struct for use
    d3dpp.Windowed = TRUE;    // program windowed, not fullscreen
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;    // discard old frames
    d3dpp.hDeviceWindow = hWnd;    // set the window to be used by Direct3D

    // create a device class using this information & information from the d3dpp stuct
    d3d->CreateDevice (D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
      D3DCREATE_SOFTWARE_VERTEXPROCESSING, &d3dpp, &d3ddev);

    return;
}
```

3. La fonction de rendu

- Effacer le contenu actuel

  HRESULT IDirect3DDevice9::Clear(DWORD count, const D3DRECT *pRects, DWORD flags, D3DCOLOR color, float z, DWORD stencil)

- Indiquer au périphérique le début de l'opération de rendu

  HRESULT IDirect3DDevice9:: BeginScene();

- Faire le rendu

- Indiquer au périphérique la fin de l'opération de rendu

  HRESULT IDirect3DDevice9:: EndScene()

- Affiche la scène.

  HRESULT IDirect3DDevice9::Present(CONST RECT *pSourceRect, CONST RECT *pDestRect, HWND hDestWindowOverride, CONST RGNDATA *pDirtyRegion);

3. La fonction de rendu

```
// this is the function used to render a single frame
void render_frame(void)
{
    // clear the window to a deep blue
    d3ddev->Clear(0, NULL, D3DCLEAR_TARGET,
   D3DCOLOR_XRGB(0, 40, 100), 1.0f, 0);

    d3ddev->BeginScene();    // begins the 3D scene

    // do 3D rendering on the back buffer here

    d3ddev->EndScene();    // ends the 3D scene

    // displays the created frame
    d3ddev->Present(NULL, NULL, NULL, NULL);
    return;
}
```

4. La fermeture de Direct3D

```
// this is the function that cleans up
// Direct3D and COM
void cleanD3D(void)
{
    // close and release the 3D device
    d3ddev->Release();

    // close and release Direct3D
    d3d->Release();

    return;
}
```

Un exemple complet

```
// include the basic windows header files
// and the Direct3D header file
#include <windows.h>
#include <windowsx.h>
#include <d3d9.h>

// include the Direct3D Library file
#pragma comment (lib, "d3d9.lib")

// global declarations
LPDIRECT3D9 d3d;    // the pointer to our Direct3D interface
LPDIRECT3DDEVICE9 d3ddev;    // pointer to the device class

// function prototypes
void initD3D(HWND hWnd);    // sets up and init Direct3D
void render_frame(void);    // renders a single frame
void cleanD3D(void);    // closes Direct3D and releases memory

// the WindowProc function prototype
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam);
```

```
// the entry point for any Windows program
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{   HWND hWnd;
    WNDCLASSEX wc;
    ZeroMemory(&wc, sizeof(WNDCLASSEX));

    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = (WNDPROC)WindowProc;
    wc.hInstance = hInstance;
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
    wc.lpszClassName = L"WindowClass";

    RegisterClassEx(&wc);
    hWnd = CreateWindowEx(NULL, L"WindowClass",
        L"Our First Direct3D Program", WS_OVERLAPPEDWINDOW,
        300, 300, 640, 480, NULL, NULL, hInstance, NULL);
    ShowWindow(hWnd, nCmdShow);
```

```
    // set up and initialize Direct3D
    initD3D(hWnd);

    // enter the main loop:

    MSG msg;

    while(TRUE)
        {
        case WM_DESTROY:
            {
                PostQuitMessage(0);
                return 0;
            } break;
    }

    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

```
// this is the main message handler for the program
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_DESTROY:
            {
                PostQuitMessage(0);
                return 0;
            } break;
    }

    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

```
// this function initializes and prepares Direct3D for use
void initD3D(HWND hWnd)
{
    d3d = Direct3DCreate9(D3D_SDK_VERSION);    // create the Direct3D interface

    D3DPRESENT_PARAMETERS d3dpp;    // a struct to hold device information

    ZeroMemory(&d3dpp, sizeof(d3dpp));    // clear out the struct for use
    d3dpp.Windowed = TRUE;    // program windowed, not fullscreen
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;    // discard old frames
    d3dpp.hDeviceWindow = hWnd;    // set the window to be used by Direct3D


    // create a device class using this information and the info from the d3dpp stuct
    d3d->CreateDevice(D3DADAPTER_DEFAULT,
                      D3DDEVTYPE_HAL,
                      hWnd,
                      D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                      &d3dpp,
                      &d3ddev);

    return;
}
```

```
// this is the function used to render a single frame
void render_frame(void)
{
    // clear the window to a deep blue
    d3ddev->Clear(0, NULL, D3DCLEAR_TARGET,
                          D3DCOLOR_XRGB(0, 40, 100), 1.0f, 0);

    d3ddev->BeginScene();    // begins the 3D scene

    // do 3D rendering on the back buffer here

    d3ddev->EndScene();    // ends the 3D scene

    // displays the created frame on the screen
    d3ddev->Present(NULL, NULL, NULL, NULL);
    return;
}
```

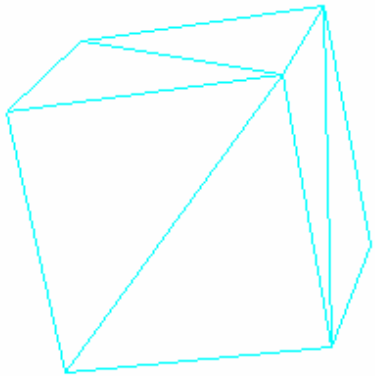
```
// this is the function that cleans up Direct3D and COM
void cleanD3D(void)
{
    d3ddev->Release();    // close and release the 3D device
    d3d->Release();    // close and release Direct3D

    return;
}
```

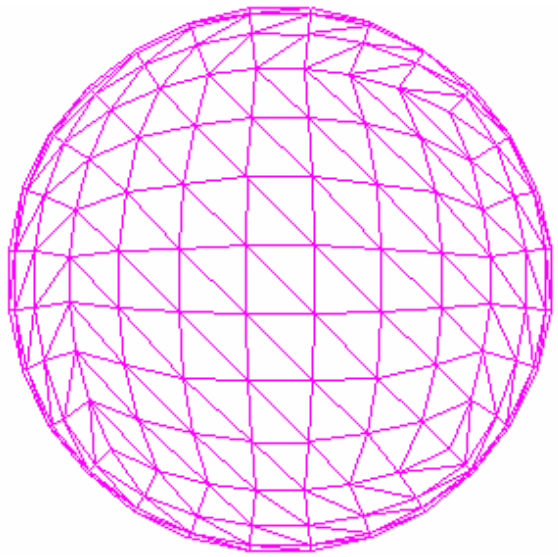
V. Maillages avec Direct3D

Plan

- Maillages
- Primitives et leurs combinaisons
- Couleurs
- Lumières
- Le tracé
  - Format des sommets
  - Tracer des triangles
  - Générer et tracer des cylindres, des spheres, etc. en utilisant des fonctions de Direct3D
- Maillages contenus dans des fichiers .x (le format de fichiers de DirectX) :
  - Chargement,
  - Extractions des matériaux
  - Affichage
  - Maillages texturés

Maillages

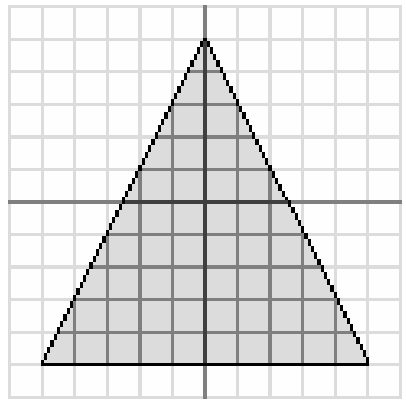




Un maillage = un ensemble triangles + données de couleurs, lumières, textures, etc.

Un triangle = 3 points

x = 0, y = 5, z = 1
x = 5, y = -5, z = 1
x = -5, y = -5, z = 1



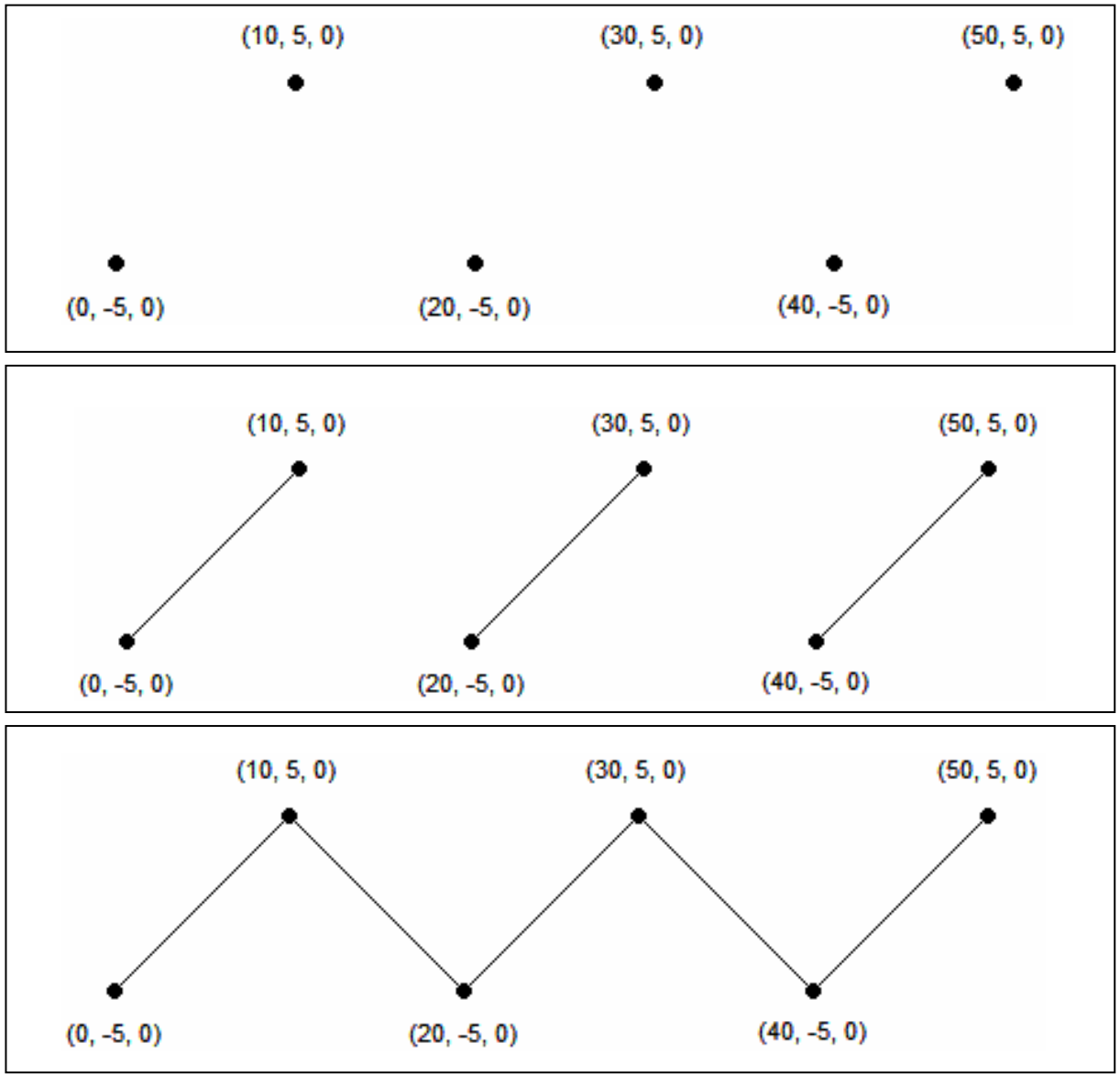
Combinaison des primitives

- Une primitive est un simple élément 3D (un point, une ligne, un triangle, etc.)

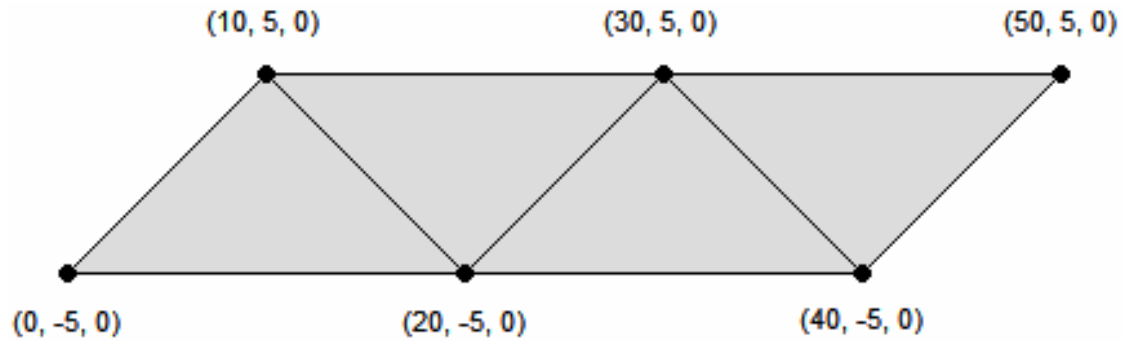
- Les primitives peuvent être combinées de plusieurs façons pour créer des objets 3D.

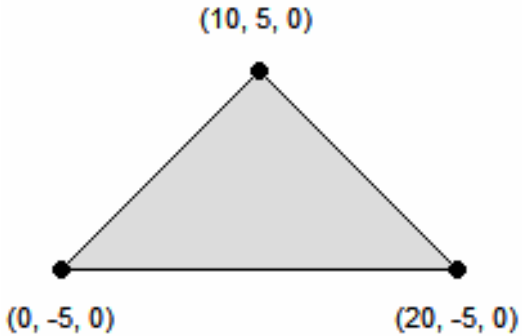
- Combinaison de primitives :
  - Point Lists
  - Line Lists
  - Line Strips
  - Triangle Lists
  - Triangle Strips
  - Triangle Fans

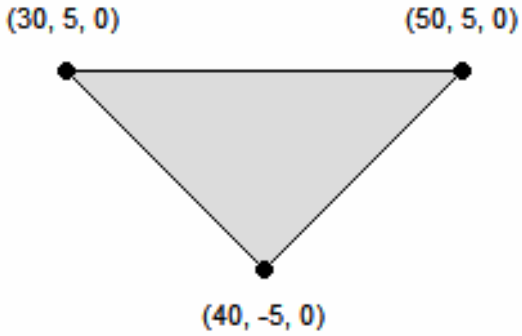
Point lists, Line lists & Line strips

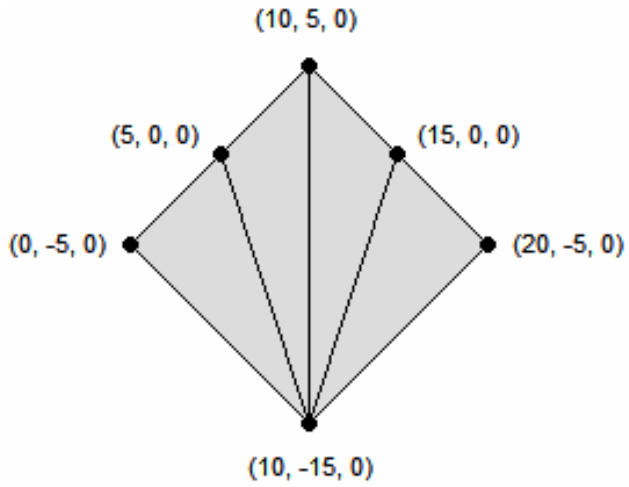


Triangle lists, triangle strips & Triangle fans



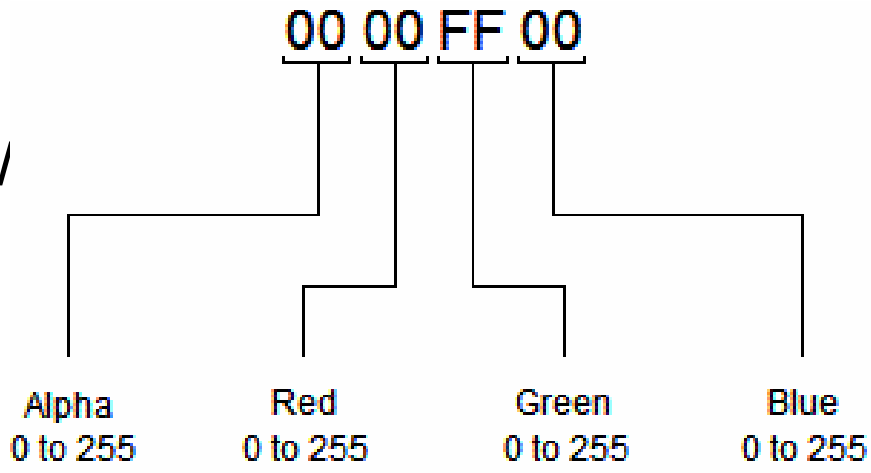






Les couleurs

- Des variables de types DW
  format RGB+Alpha :



- Exemples :
  - DWORD Color_A = 0xff00ff00;
  - DWORD Color_B = 0x88ff00cc;
  - DWORD Color_A = D3DCOLOR_XRGB(0, 255, 0);
  - DWORD Color_B = D3DCOLOR_ARGB(136, 255, 0, 204);

Lumière & éclairage

- Chaque objet réagit à la lumière en fonction des propriétés matérielles de sa surface.

- Un objet peut émettre une lumière propre, renvoyer dans toutes les directions la lumière qu'il reçoit, ou réfléchir une partie de la lumière dans une direction particulière, comme un miroir ou une surface brillante.

- Quatre types de lumières: emise, ambiante, diffuse ou spéculaire.

- Lumière émise : Ne concerne que les objets
  Les objets peuvent émettre une lumière propre, qui augmentera leur intensité, mais n'affectera pas les autres objets de la scène.

- Lumière ambiante : Concerne les objets et les lampes
  C'est la lumière qui a tellement été dispersée et renvoyée par l'environnement qu'il est impossible de déterminer la direction d'où elle émane. Elle semble venir de toutes les directions. Quand une lumière ambiante rencontre une surface, elle est renvoyée dans toutes les directions.

Lumière & éclairage

- Lumière diffuse : Concerne les objets et les lampes

  C'est la lumière qui vient d'une direction particulière, et qui va être plus brillante si elle arrive perpendiculairement à la surface que si elle est rasante. Par contre, après avoir rencontré la surface, elle est renvoyée uniformément dans toutes les directions.

- Lumière spéculaire : Concerne les objets et les lampes

  La lumière spéculaire vient d'une direction particulière et est renvoyée par la surface dans une direction particulière. Par exemple un rayon laser réfléchi par un miroir.

- Brillance : Ne concerne que les objets

  Cette valeur entre 0.0 et 128.0 détermine la taille et l'intensité de la tâche de réflexion spéculaire. Plus la valeur est grande, et plus la taille est petite et l'intensité importante.

Le tracé

- Sommets et vertex Buffers
- Comment tracer un triangle
- Générer et tracer des cylindres, des sphères, etc. en utilisant des fonctions de Direct3D

Format des sommets

- Un sommet = coordonnées géométriques + propriétés associées

- DirectX code les sommets avec une technologie dite Flexible Vertex Format (or FVF):

```
#define CUSTOMFVF (D3DFVF_XYZRHW | D3DFVF_DIFFUSE)
struct CUSTOMVERTEX
    {
        FLOAT x, y, z, rhw;    // from the D3DFVF_XYZRHW flag
        DWORD color;    // from the D3DFVF_DIFFUSE flag
    }
```

- Dans la structure CUSTOMVERTEX les 4 premiers attributs FLOAT sont représentés par le flag D3DFVF_XYZRHW, le dernier attribut de type DWORD est représenté par le flag D3DFVF_DIFFUSE.

- Le #define fait le lien entre la structure du programmeur et les éléments de Direct3D.

Créer des sommets

- Créer un sommet :

```
CUSTOMVERTEX OurVertex = {320.0f, 50.0f, 1.0f, 1.0f,
    3DCOLOR_XRGB(0, 0, 255)};
```

- Créer un tableau de sommets :

```
CUSTOMVERTEX OurVertices[] =
{
    {320.0f, 50.0f, 1.0f, 1.0f, D3DCOLOR_XRGB(0, 0, 255),},
    {520.0f, 400.0f, 1.0f, 1.0f, D3DCOLOR_XRGB(0, 255, 0),},
    {120.0f, 400.0f, 1.0f, 1.0f, D3DCOLOR_XRGB(255, 0, 0),},
};
```

Vertex Buffer pour stocker les sommets

- Une fonction du périphérique d3d pour créer les Vertex Buffers :

```
HRESULT CreateVertexBuffer(
        UINT Length, DWORD Usage, DWORD FVF, D3DPOOL Pool,
        LPDIRECT3DVERTEXBUFFER9 ppVertexBuffer,
        HANDLE* pSharedHandle);
```

- Exemple d'utilisation :

```
LPDIRECT3DVERTEXBUFFER9 t_buffer;

d3ddev->CreateVertexBuffer(3*sizeof(CUSTOMVERTEX),
                           0,
                           CUSTOMFVF,
                           D3DPOOL_MANAGED,
                           &t_buffer,
                           NULL);
```

Vertex Buffer pour stocker les sommets

- Avant de remplir le Vertex Buffer avec les sommets, il faut le verrouiller :

  HRESULT Lock(UINT OffsetToLock, UINT SizeToLock,
  VOID** ppbData, DWORD Flags);

- Exemple d'utilisation

```
VOID* pVoid;
// verrouiller t_buffer,
t_buffer->Lock(0, 0, (void**)&pVoid, 0);
// copier les sommets dand le buffer avec memcopy
memcpy(pVoid, OurVertices, sizeof(OurVertices));
// Déverrouiller le buffer
t_buffer->Unlock();    // unlock t_buffer
```

Récapitulation sur les sommets

```
void init_graphics(void)
    {
        // create three vertices using the CUSTOMVERTEX struct built earlier
        CUSTOMVERTEX t_vert[] =
        {
            { 320.0f, 50.0f, 0.5f, 1.0f, D3DCOLOR_XRGB(0, 0, 255), },
            { 520.0f, 400.0f, 0.5f, 1.0f, D3DCOLOR_XRGB(0, 255, 0), },
            { 120.0f, 400.0f, 0.5f, 1.0f, D3DCOLOR_XRGB(255, 0, 0), },
        };

        // create the vertex and store the pointer into t_buffer, which is created globally
        d3ddev->CreateVertexBuffer(3*sizeof(CUSTOMVERTEX),
                                   0,
                                   CUSTOMFVF,
                                   D3DPOOL_MANAGED,
                                   &t_buffer,
                                   NULL);

        VOID* pVoid;    // the void pointer

        t_buffer->Lock(0, 0, (void**)&pVoid, 0);    // lock the vertex buffer
        memcpy(pVoid, t_vert, sizeof(t_vert));    // copy the vertices to the locked buffer
        t_buffer->Unlock();    // unlock the vertex buffer

        return;
    }
```

Afficher le triangle

- Pour afficher le triangle contenu dans le vertex buffer, nous avons trois fonctions à appeler à partir du périphérique associé à Direct3d

  – SetVFF() : quel VFF code utilisons-nous?

    d3ddev->SetFVF(CUSTOMFVF);

  – SetStreamSource() : quel vertex buffer utilisons-nous?.

    HRESULT SetStreamSource(UINT StreamNumber,
    LPDIRECT3DVERTEXBUFFER9 pStreamData,
    UINT OffsetInBytes,
    UINT Stride);

    d3ddev->SetStreamSource(0, t_buffer, 0, sizeof(CUSTOMVERTEX));

  – DrawPrimitive() : affichage

    HRESULT DrawPrimitive(D3DPRIMITIVETYPE PrimitiveType,
    UINT StartVertex, UINT PrimitiveCount);

Afficher le triangle

```
void render_frame(void)
{
    d3ddev->Clear(0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0);
    d3ddev->BeginScene();

        // select which vertex format we are using
        d3ddev->SetFVF(CUSTOMFVF);

        // select the vertex buffer to display
        d3ddev->SetStreamSource(0, t_buffer, 0, sizeof(CUSTOMVERTEX));

        // copy the vertex buffer to the back buffer
        d3ddev->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);

    d3ddev->EndScene();
    d3ddev->Present(NULL, NULL, NULL, NULL);
    return;
}
```

- **Libérer le buffer lorsque nous en avons plus besoin :**

```
void cleanD3D(void) {
    t_buffer->Release();    // close and release the vertex buffer
    d3ddev->Release();    // close and release the 3D device
    d3d->Release();        // close and release Direct3D
    return;
}
```

Fonctions Direct3D pour générer des maillages

- Création : Plusieurs fonctions de création
  - Teapot : D3DXCreateTeapot
  - Solid Rectangulaire (cube) : D3DXCreateBox
  - Cylindre : D3DXCreateCylinder
  - Sphere : D3DXCreateSphere
  - …
- Affichage
  - Fonction DrawSubset()

- Fermeture
  - Fonction Release()

Teapot

- Création :

  HRESULT D3DXCreateTeapot(LPDIRECT3DDEVICE9 pDevice, LPD3DXMESH *ppMesh, LPD3DXBUFFER *ppAdjacency);
  - pDevice : pointeur vers le périphérique Direct3D
  - ppMesh : pointeur vers un pointeur sur le maillage
  - ppAdjacency : topologie et adjacence. Mettre a NULL
- Exemple :

  LPD3DXMESH meshTeapot; // define the mesh pointer

  // create the teapot

  D3DXCreateTeapot(d3ddev, &meshTeapot, NULL);

  // Dans la fonction de rendu, draw the teapot

  meshTeapot->DrawSubset(0);

  // Dans la fonction CleanD3d

  meshTeapot->Release(); // close and release the teapot mesh

Solide rectangulaire

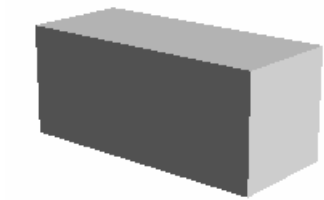
- Création :

  HRESULT D3DXCreateBox(LPDIRECT3DDEVICE9 pDevice, FLOAT Width, FLOAT Height, FLOAT Depth, LPD3DXMESH *ppMesh, LPD3DXBUFFER *ppAdjacency);

- Exemple :

  LPD3DXMESH meshBox; // define the mesh pointer

  D3DXCreateBox(d3ddev, 7.0f, 3.0f, 3.0f, &meshBox, NULL); // create a box



Cylindre

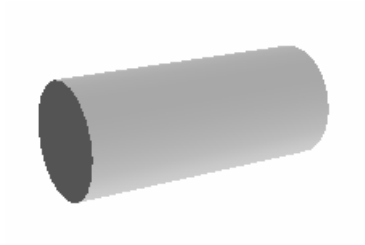
- Création :

  RESULT D3DXCreateCylinder(LPDIRECT3DDEVICE9 pDevice, FLOAT Radius1, FLOAT Radius2, FLOAT Length, UINT Slices, UINT Stacks, LPD3DXMESH *ppMesh, LPD3DXBUFFER *ppAdjacency);

- Exemple :

  LPD3DXMESH meshCylinder; // define the mesh

  pointerD3DXCreateCylinder(d3ddev, 1.5f, 1.5f, 7.0f, 20, 10, &meshCylinder, NULL);



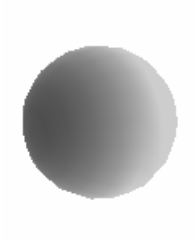
Sphère

- Création

  RESULT D3DXCreateSphere(LPDIRECT3DDEVICE9 pDevice, FLOAT Radius, UINT Slices, UINT Stacks, LPD3DXMESH *ppMesh, LPD3DXBUFFER *ppAdjacency);

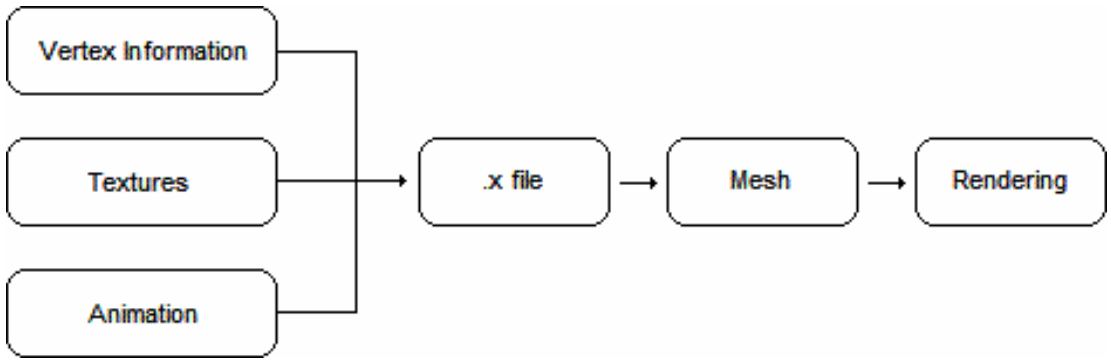
- Exemple

```
LPD3DXMESH meshSphere;    // define the mesh pointer
// create a sphere
D3DXCreateSphere(d3ddev, 2.0f, 20, 10, &meshSphere, NULL);
```



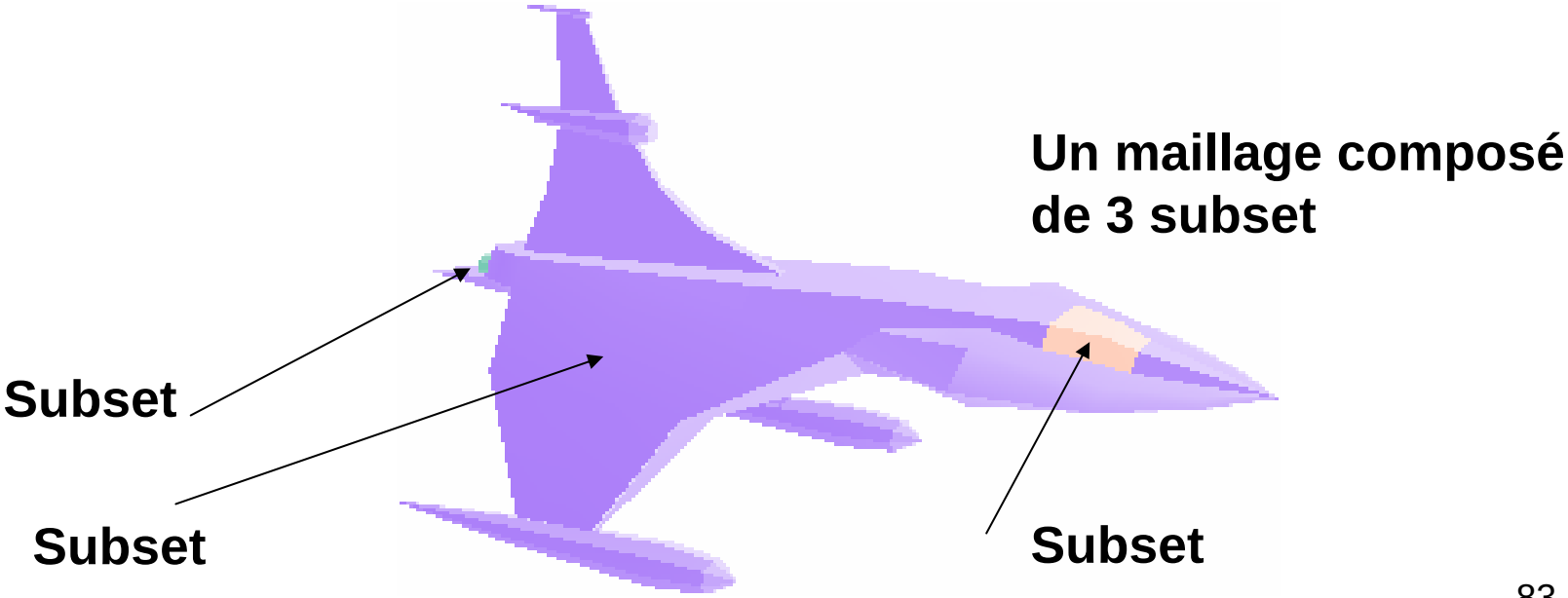
Maillages contenus dans des fichiers .x

- Un maillage (mesh) est un fichier contenant un ensemble d'informations sur un modèle 3D : les sommets, les textures, les animations, etc.
- Direct3D utilise le format .x pour représenter les maillages
- Il existe un grand nombre d'outils pour générer, importer, ou exporter un modèle 3D vers ce format.
- On peut par exemple modéliser une forme avec 3DSMax ou Maya puis l'exporter en .x



Maillages à partir d'un fichier .x

- 3 étapes
  - Charger le maillage (le fichier .x)
  - Extraire les informations utiles
    - les sous-maillages = subset,
    - les couleurs,
    - Les matériaux
  - Afficher les différents subsets



Charger le maillage

```
HRESULT D3DXLoadMeshFromX(LPCTSTR pFilename,
                          DWORD Options,
                          LPDIRECT3DDEVICE9 pD3DDevice,
                          LPD3DXBUFFER *ppAdjacency,
                          LPD3DXBUFFER *ppMaterials,
                          LPD3DXBUFFER *ppEffectInstances,
                          DWORD *pNumMaterials,
                          LPD3DXMESH *ppMesh);
```

- pFilename: nom du fichier .x
- Options: une combinaison de paramètres indiquant comment sera géré le maillage. Pour stocker le maillage dans la memoire utiliser l'option: D3DXMESH_SYSTEMMEM
- ppMaterials: stocke les informations sur les matériaux
- ppEffectIntces: les effets particuliers. NULL
- ppAdjacency: Les adjacences. NULL
- ppMesh: un pointeur sur le maillage

Charger le maillage

- **Exemple :**

```
// global variables
LPD3DXMESH meshSpaceship;
DWORD numMaterials;

// local variable (in the init_graphics() function)
LPD3DXBUFFER bufShipMaterial;

D3DXLoadMeshFromX(L"spaceship 2.x",    // load this file
                  D3DXMESH_SYSTEMMEM,    // load the mesh into
                                         // system memory
                  d3ddev,    // the Direct3D Device
                  NULL,    // we aren't using adjacency
                  &bufShipMaterial,    // put the materials here
                  NULL,    // we aren't using effect instances
                  &numMaterials,    // the number of materials in this model
                  &meshSpaceship);    // put the mesh here
```

Tracer le maillage chargé? Non pas encore!

- A ce stade on peut tracer le maillage chargé, mais on aura un modèle gris sans aucune couleur ou texture.

- Pour avoir les couleurs, et un meilleur aspect visuel, il faut lire les matériaux à partir du 5e paramètre de la fonction D3DXLoadMeshFromX.

Extraire les matériaux

```
D3DMATERIAL9* materal;    // a pointer to a material buffer

// retrieve the pointer to the buffer containing the material information
D3DXMATERIAL* tempMaterials =
(D3DXMATERIAL*)bufShipMaterial->GetBufferPointer();

// create a new material buffer for each material in the mesh
material = new D3DMATERIAL9[numMaterials];

for(DWORD i = 0; i < numMaterials; i++)    // for each material...
{
    material[i] = tempMaterials[i].MatD3D;    // get the material info...
    material[i].Ambient = material[i].Diffuse;    // and make ambient the
same as diffuse
}
```

Tracer

- Pour chaque subset
  - Sélectionner ses matériaux
  - Le tracer

- Exemple :

```
// loop through each subset
for(DWORD i = 0; i < numMaterials; i++)    {
    // set the appropriate material for the subset
    d3ddev->SetMaterial(&material[i]);
    // draw the subset
    meshSpaceship->DrawSubset(i);
}
```

Charger un maillage texturé

- 2 étapes :
  - Charger la texture lorsqu'on charge le maillage
  - Sélectionner la texture de chaque subset avant de le tracer.

Exemple chargement de texture

```
// a pointer to a material buffer
D3DMATERIAL9* materal;
// a pointer to a texture
LPDIRECT3DTEXTURE9* texture;

// retrieve the pointer to the buffer containing the
//material information
D3DXMATERIAL* tempMaterials =
(D3DXMATERIAL*)bufShipMaterial->GetBufferPointer();

// create a new material buffer and texture for each material in
// the mesh
material = new D3DMATERIAL9[numMaterials];
texture = new LPDIRECT3DTEXTURE9[numMaterials];
```

Exemple chargement de texture

```
for(DWORD i = 0; i < numMaterials; i++)    // for each material...
    {
        material[i] = tempMaterials[i].MatD3D;    // get the material info...

    // and make ambient the same as diffuse
    material[i].Ambient = material[i].Diffuse;
    USES_CONVERSION;    // allows certain string conversions

    // if there is a texture to load, load it
        if(FAILED(D3DXCreateTextureFromFile(
                d3ddev,
                CA2W(tempMaterials[i].pTextureFilename),
                &texture[i]))
            )
        // if there is no texture, set the texture to NULL
        texture[i] = NULL;
}
```

Sélectionner les textures (comme matériaux)

```
 // draw the spaceship
 // loop through each subset
for(DWORD i = 0; i < numMaterials; i++)
{
    // set the material for the subset
   d3ddev->SetMaterial(&material[i]);

    // if the subset has a texture (if texture is not NULL)
    if(texture[i] != NULL)
        d3ddev->SetTexture(0, texture[i]);    // ...then set the texture

    meshSpaceship->DrawSubset(i);    // draw the subset
}
```

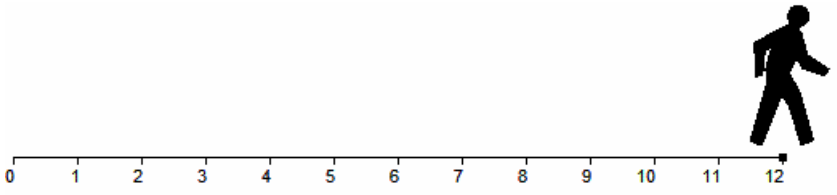
VI. Bases de la visualisation en 3D

Plan

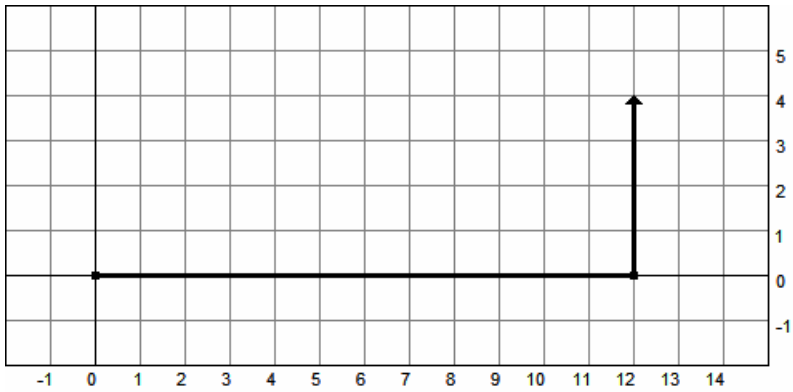
- Systèmes de coordonnées
- Les transformations
  - Transformations de modélisation
  - Transformations de visualisation
  - Transformations de projection
- Le Z-buffer

Système de coordonnées

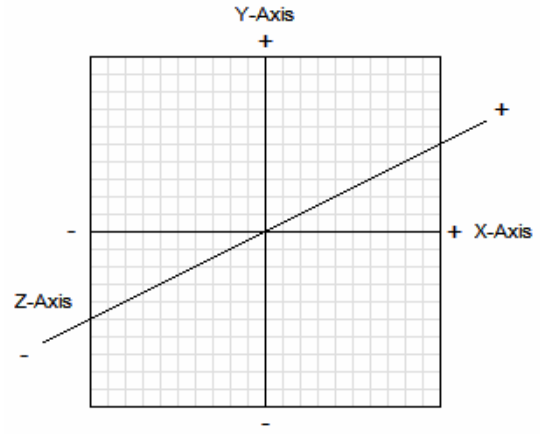
- 1 D



- 2 D



- 3 D

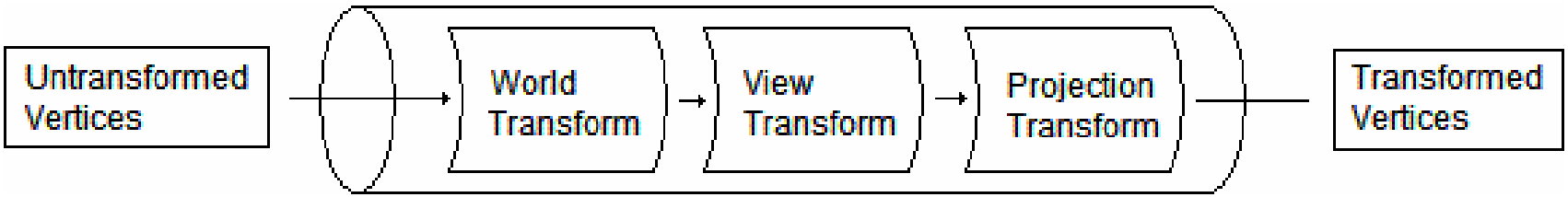


The geometry pipeline

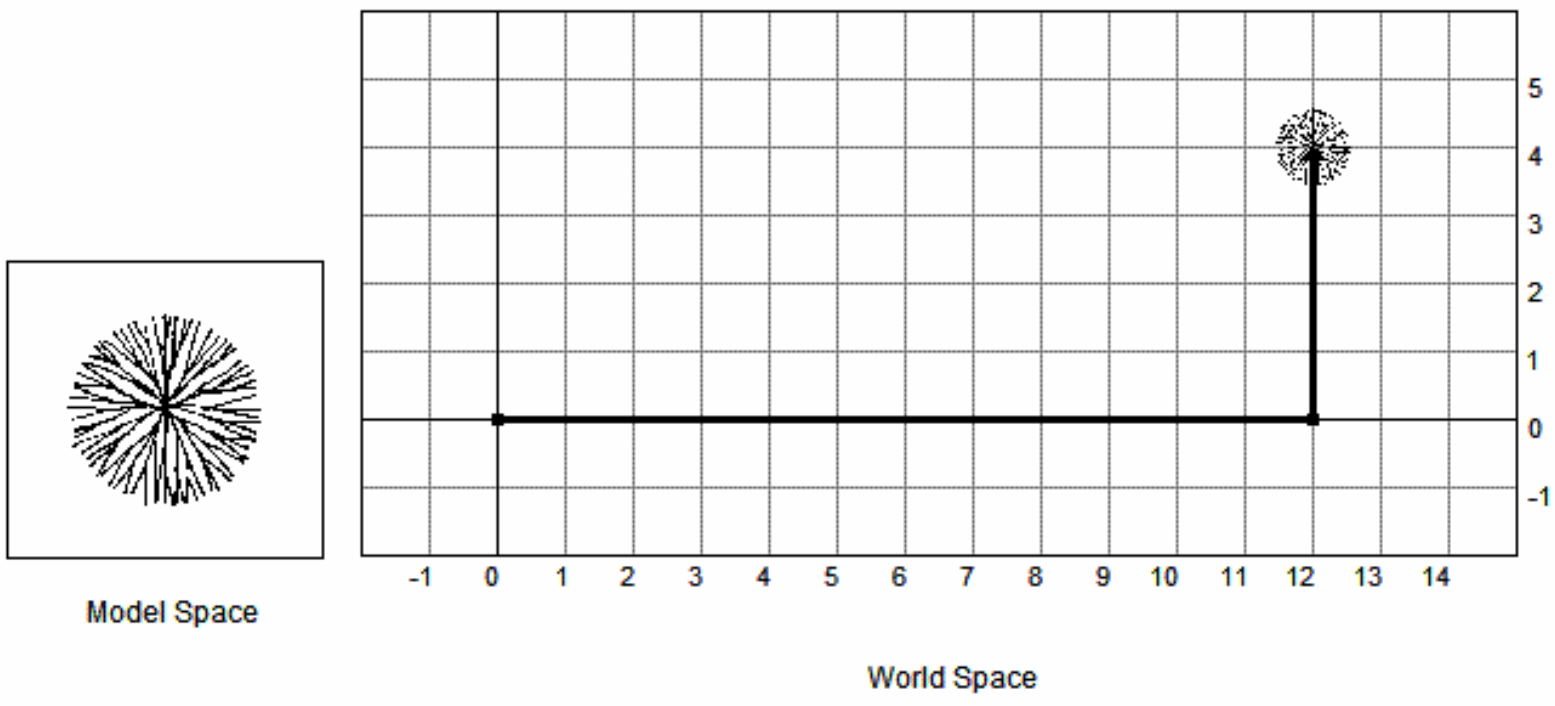
Q: Comment convertir un objet 3D en une image 2D visible à l'écran?

R: en appliquant un ensemble de transformations :

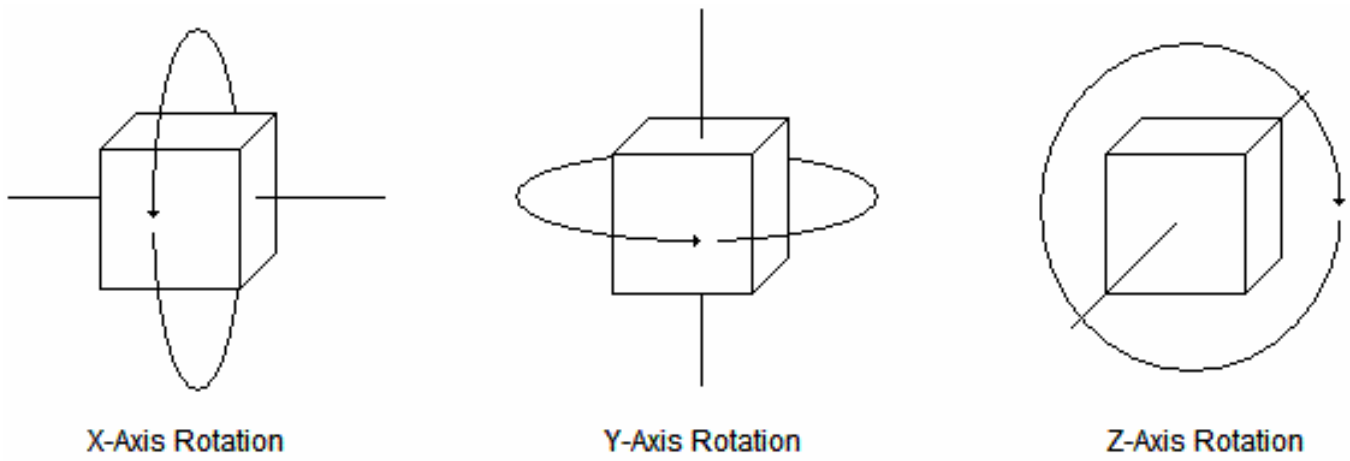
- Transformation de modélisation :
  - Translation
  - Rotation
  - Mise à l'échelle (réduction/agrandissement)
- Transformation de visualisation
- Transformation de projection



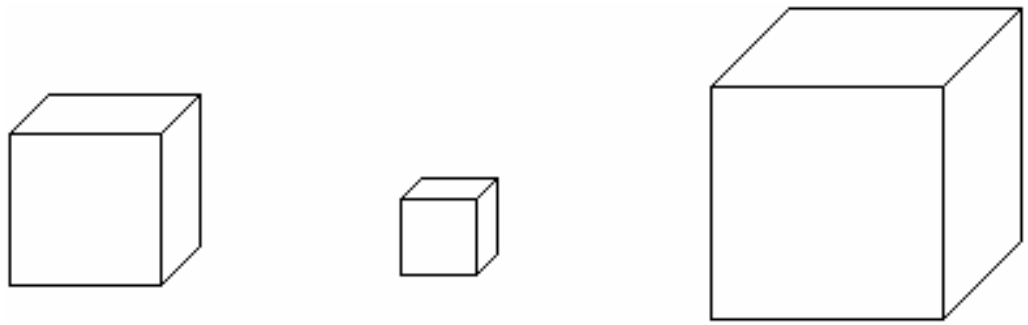
Transformation de modélisation : Translation



Transformation de modélisation : Rotation



Transformation de modélisation : dimensionnement



Toutes ces transformations sont codées et réalisées par des matrices.
Comment coder et manipuler les matrices (addition, multiplication, etc.) sous directX?

Matrices de transformation sous DirectX

- Une matrice :

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

```
float TheMatrix [4][4] =
{
    1.0f,   2.0f,   3.0f,   4.0f,
    5.0f,   6.0f,   7.0f,   8.0f,
    9.0f,   10.0f, 11.0f, 12.0f,
    13.0f, 14.0f, 15.0f, 16.0f
};
```

- Le type matrice

```
typedef struct _D3DMATRIX {
    union {
        struct {
        float       _11, _12, _13, _14;
        float       _21, _22, _23, _24;
        float       _31, _32, _33, _34;
        float       _41, _42, _43, _44;
        };
        float m[4][4];
    }
} D3DMATRIX;
```

Matrices de transformation sous DirectX

- Initialisation matrice unité :

```
D3DXMATRIX* D3DXMatrixIdentity(D3DXMATRIX* pOut);
```

- Translation :

```
D3DXMATRIX matTranslate;    // to store the translation information

// build a matrix to move the model 12 units along the x-axis
// and 4 units along the y-axis store it to matTranslate
D3DXMatrixTranslation(&matTranslate, 12.0f, 4.0f, 0.0f);

// tell Direct3D about our matrix
d3ddev->SetTransform(D3DTS_WORLD, &matTranslate);
```

Matrices de transformation sous DirectX

- Rotation :

```
D3DXMATRIX matRotateX;    // a matrix to store the rotation information

// build a matrix to rotate the model 3.14 radians
D3DXMatrixRotationX(&matRotateX, 3.14f);

// tell Direct3D about our matrix
d3ddev->SetTransform(D3DTS_WORLD, &matRotateX);
```

- L'angle de rotation peut aussi être donné en degré :

```
D3DXMatrixRotationX(&matRotateX, D3DXToRadian(180.0f));
```

- Mise à l'échelle :

```
D3DXMATRIX matScale;    // a matrix to store the scaling information

// build a matrix to double the size of the model store it to matScale
D3DXMatrixScaling(&matScale, 2.0f, 2.0f, 2.0f);

// tell Direct3D about our matrix
d3ddev->SetTransform(D3DTS_WORLD, &matScale);
```

Combiner plusieurs transformations

```
D3DXMATRIX matRotateX;    // to store the rotation information
D3DXMATRIX matScale;    // a matrix to store the scaling information

// double the size of the model
D3DXMatrixScaling(&matScale, 2.0f, 2.0f, 2.0f);
// rotate the model 90 degrees
D3DXMatrixRotationX(&matRotateX, D3DXToRadian(90.0f));

// set the world transform to the two matrices multiplied together
d3ddev->SetTransform(D3DTS_WORLD, &(matRotateX * matScale));
```

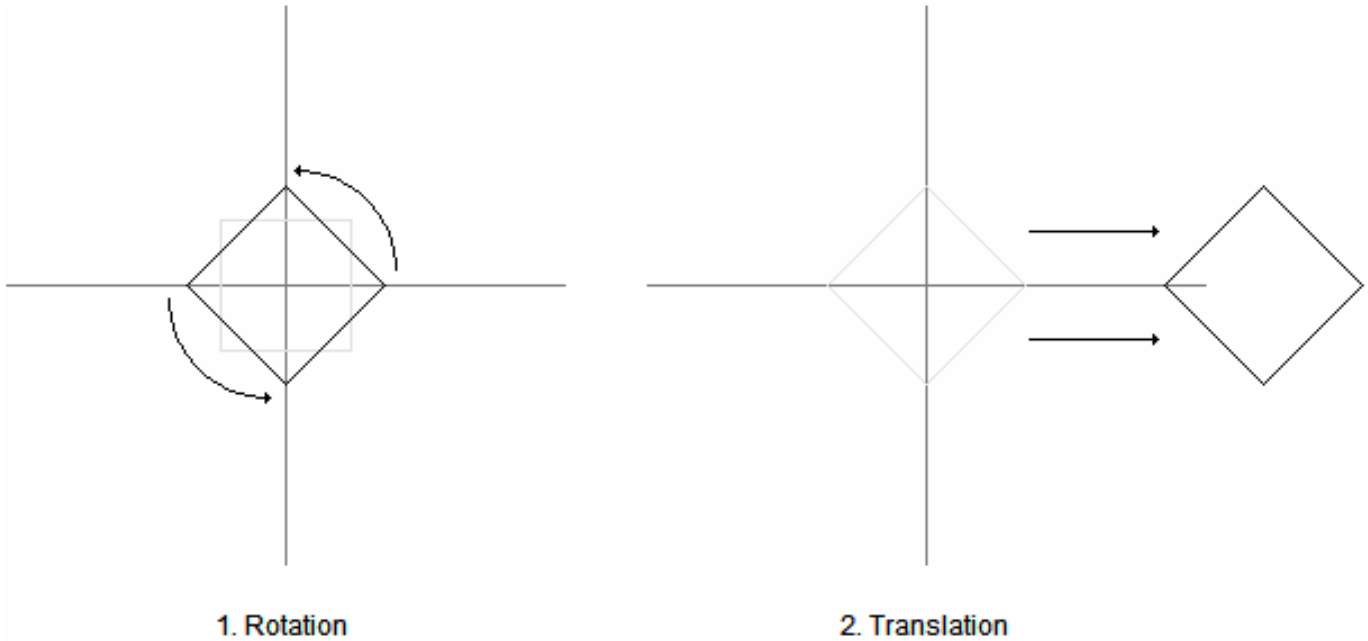
Avec ce code, tous les objets de la scène subiront une rotation de 90 degré au tour de l'axe X et seront dédoublés en taille.

Combiner plusieurs transformations

```
D3DXMATRIX matRotateX;  D3DXMATRIX matRotateY;
D3DXMATRIX matRotateZ;
D3DXMATRIX matScale;
D3DXMATRIX matTranslate;

D3DXMatrixRotationX(&matRotateX, D3DXToRadian(50.0f));
D3DXMatrixRotationY(&matRotateY, D3DXToRadian(50.0f));
D3DXMatrixRotationZ(&matRotateZ, D3DXToRadian(50.0f));
D3DXMatrixScaling(&matScale, 5.0f, 1.0f, 1.0f);
D3DXMatrixTranslation(&matTranslate, 40.0f, 12.0f, 0.0f);

d3ddev->SetTransform(D3DTS_WORLD,
                &(matRotateX * matRotateY * matRotateZ * matScale *
                   matTranslate));
```

Rotation autour de X, puis de Y, puis de Z, puis agrandissement dans le sens des X, et enfin translation

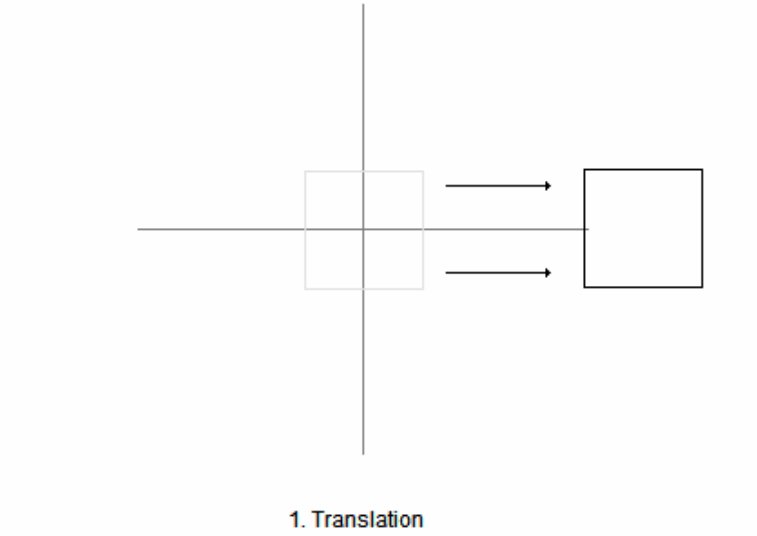
Combiner plusieurs transformations

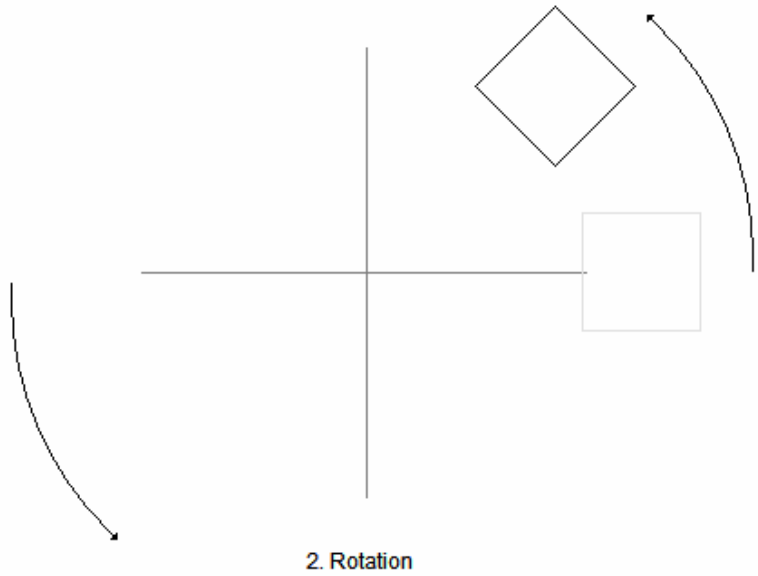


L’ordre des transformations est important, ici on fait une rotation suivie d’une translation

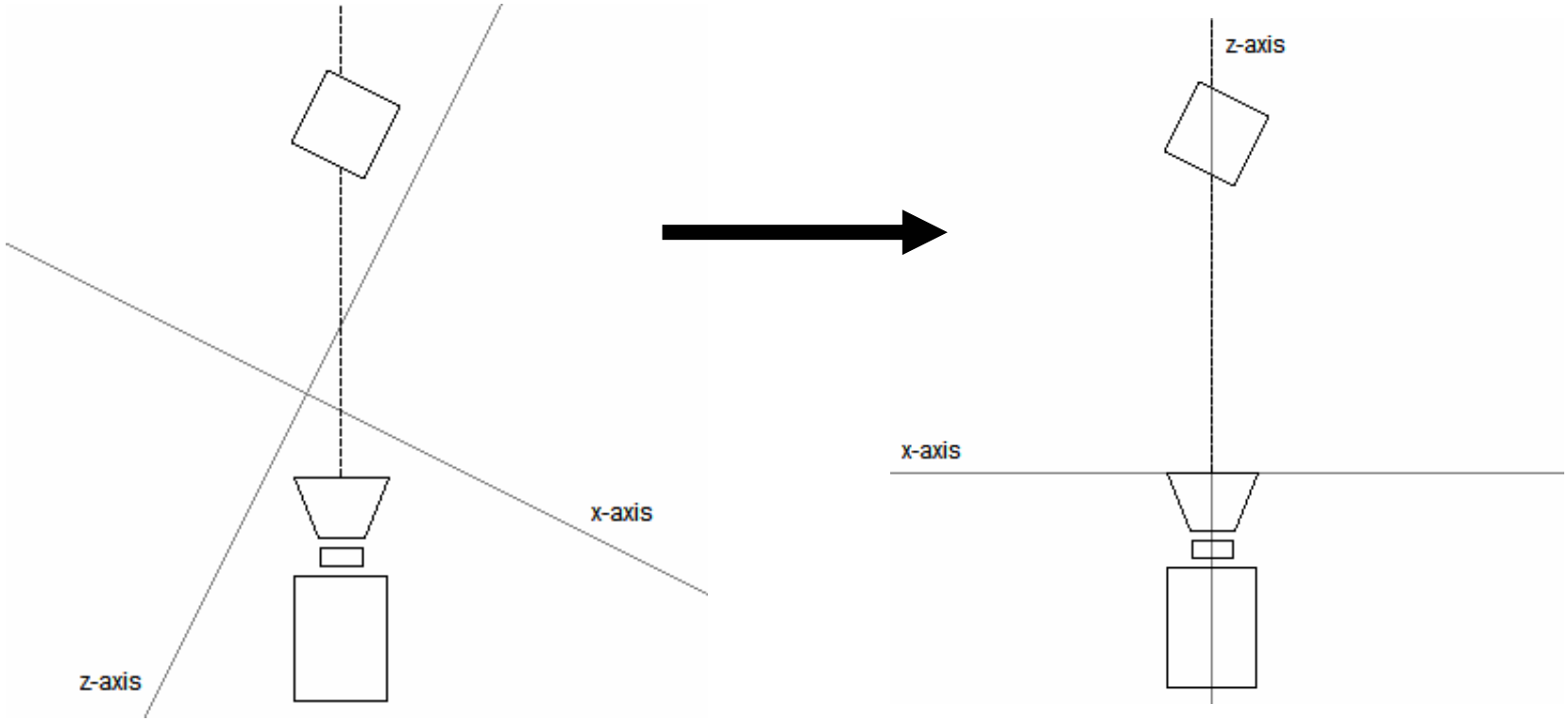
Combiner plusieurs transformations

L’ordre des transformations est important, ici on fait une translation suivie d’une rotation





Transformation de visualisation



- Changer l'origine et les orientations des axes pour mieux positionner la caméro. La caméra regarde dans la direction des Z.
- Calculer les nouvelles coordonnées géométriques des objets suite au positionnement des axes.

Transformation de visualisation

- Les transformations de visualisation sont assurées par la fonction :

```
D3DXMATRIX* D3DXMATRIXLookAtLH(D3DXMATRIX* pOut,
                               CONST D3DXVECTOR3* pEye,
                               CONST D3DXVECTOR3* pAt,
                               CONST D3DXVECTOR3* pUp);
```

- Le type D3DXVECTOR3 est défini par :

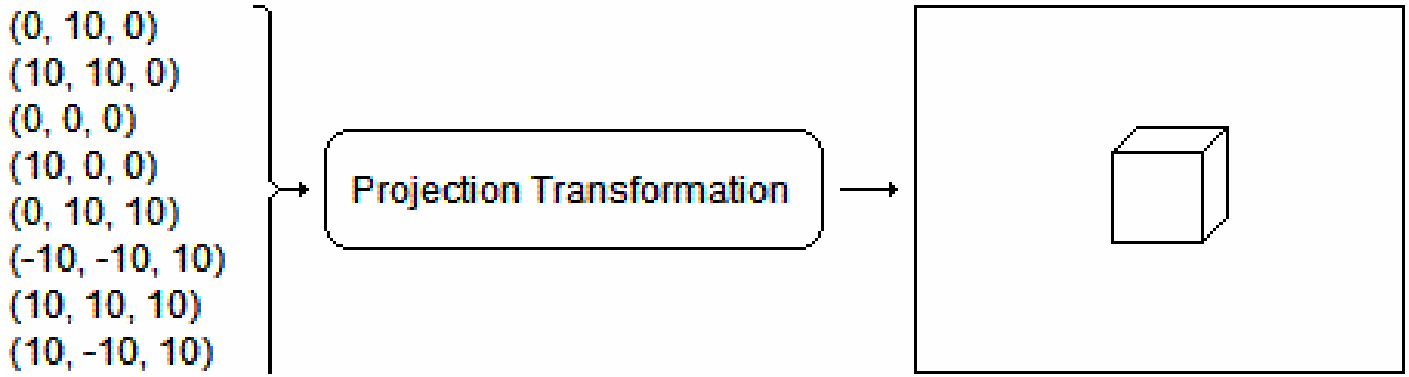
```
typedef struct D3DXVECTOR3
{
    FLOAT x;    // contains an x-axis coordinate
    FLOAT y;    // contayns a y-axis coordinate
    FLOAT z;    // containz a z-axis coordinate
} D3DXVECTOR3, *LPD3DXVECTOR3;
```

Transformation de visualisation

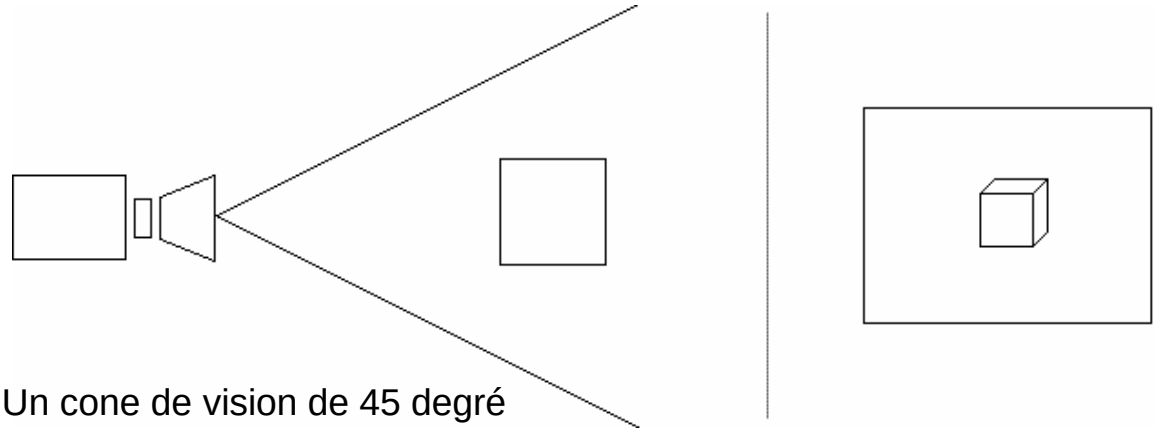
Exemple : une transformation de visualisation pour voir un objet positionné à l'origine du repère global (0, 0, 0) avec une caméra positionnée à (100, 100, 100).

```
D3DXMATRIX matView;    // the view transform matrix

D3DXMatrixLookAtLH(&matView,
        &D3DXVECTOR3 (100.0f, 100.0f, 100.0f), // the camera position
        &D3DXVECTOR3 (0.0f, 0.0f, 0.0f),    // the look-at position
        &D3DXVECTOR3 (0.0f, 1.0f, 0.0f));    // the up direction

// set the view transform to matView
d3ddev->SetTransform(D3DTS_VIEW, &matView);
```

Transformation de projection

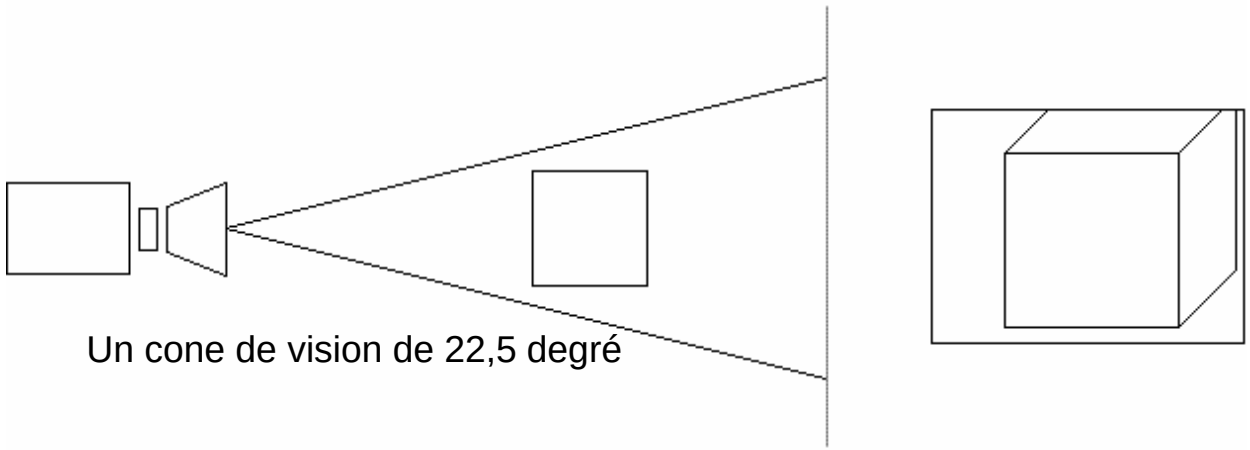


Le cone de vision

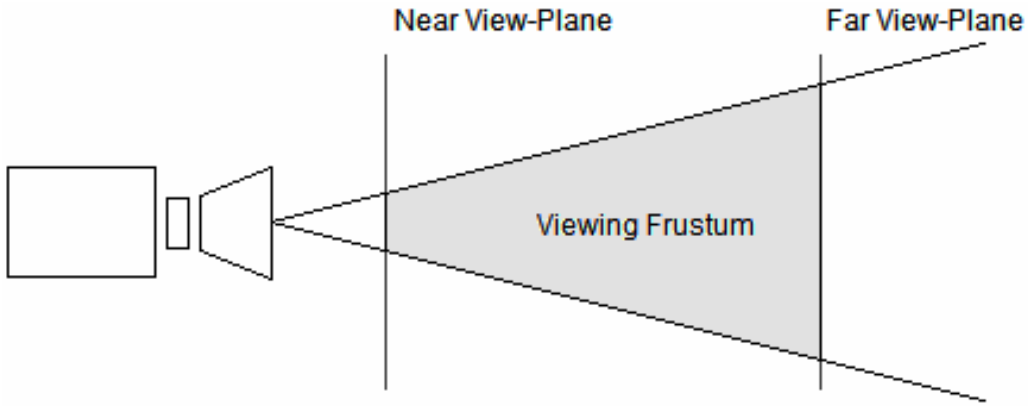


Un cone de vision de 45 degré

Transformation de projection



Le frustum de visualisation



Transformation de projection

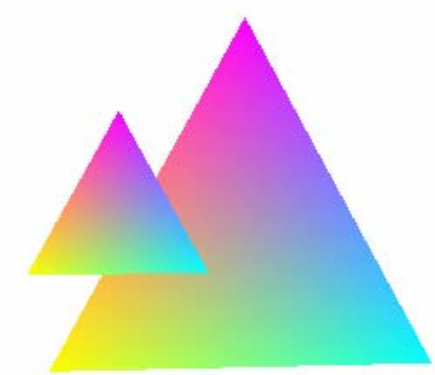
```
D3DXMATRIX* D3DXMatrixPerspectiveFovLH(D3DXMATRIX* pOut,
                    FLOAT fovy,  FLOAT Aspect, FLOAT zn, FLOAT zf);
```

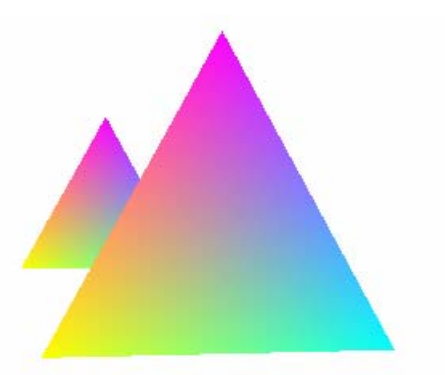
Exemple d'une transformation de projection :

```
D3DXMATRIX matProjection;    // the projection transform matrix

D3DXMatrixPerspectiveFovLH(&matProjection,
      D3DXToRadian(45),    // the horizontal field of view
      (FLOAT)SCREEN_WIDTH / (FLOAT)SCREEN_HEIGHT, // aspect ratio
      1.0f,    // the near view-plane
      100.0f);    // the far view-plane

// set the projection transform
d3ddev->SetTransform(D3DTS_PROJECTION, &matProjection);
```

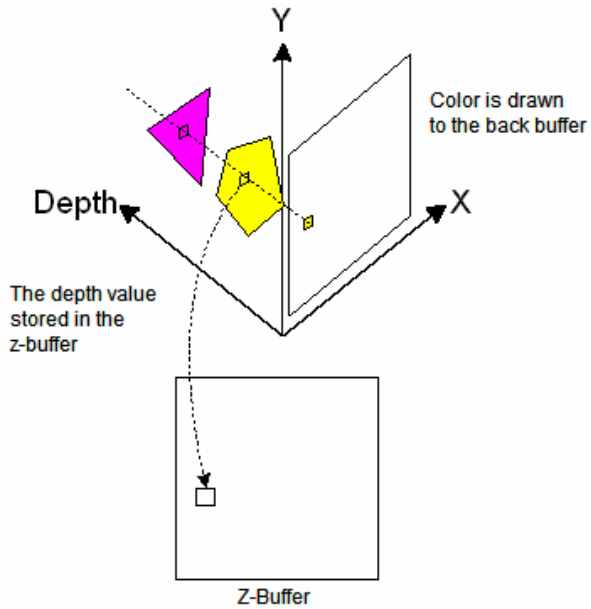
Le Z-buffer

Si le plus grand triangle est plus proche de la caméra, on doit avoir cette image au lieu de celle-ci





Le Z-buffer = c'est le pixel le plus proche de la caméra qui apparu à l'écran



Utiliser le Z-buffer

- Dire que l'on veut utiliser le Z-buffer lors de l'initialisation de Direct3d (dans initD3d()) :

```
D3DPRESENT_PARAMETERS d3dpp;
ZeroMemory(&d3dpp, sizeof(d3dpp));
d3dpp.Windowed = FALSE;
d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
d3dpp.hDeviceWindow = hWnd;
d3dpp.BackBufferFormat = D3DFMT_X8R8G8B8;
d3dpp.BackBufferWidth = SCREEN_WIDTH;
d3dpp.BackBufferHeight = SCREEN_HEIGHT;
d3dpp.EnableAutoDepthStencil = TRUE;
d3dpp.AutoDepthStencilFormat = D3DFMT_D16;
```

- Autoriser le Z-buffer (dans initD3d())

```
// turn off the 3D lighting
d3ddev->SetRenderState(D3DRS_LIGHTING, FALSE);
// turn on the z-buffer
d3ddev->SetRenderState(D3DRS_ZENABLE, TRUE);
```

- Effacer le Z-buffer (dans la fonction de visualisation render_frame())

```
d3ddev->Clear(0, NULL, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0);
d3ddev->Clear(0, NULL, D3DCLEAR_ZBUFFER, D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0);
```