GIÁO ÁN TRAINING DIRECTX 9.0

GameUIT Club

Tổng hợp các bài tutorial của các anh trên forum GameUIT.com

Xin cảm ơn các anh đã chia sẻ kiến thức và thời gian của mình cho CLB.

Contents

Phần 1: Beginning game with DirectX

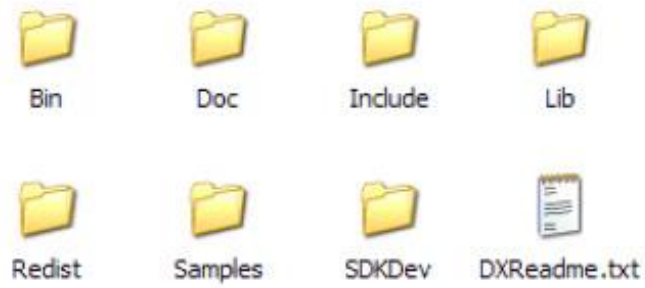
I. DirectX là gì ?

- Nói nôm na đó là một bộ thư viện cung cấp các API phục vụ cho các chức năng đa phương tiện, video, âm thanh trên nền tảng của Microsoft. Cụ thể ở đây là viết game trên windows.
- DirectX ra đời năm 1995 với tên gọi đầu tiên là The Game SDK. Kể từ ver 2.0 đã được đổi lại là DirectX. Hiện nay đã có DirectX 11 nhưng chắc còn lâu mới được học quá.

II. Cài đặt DirectX như thế nào ?

Để sử dụng DirectX bạn cần :
- Visual C++ hoặc Visual Studio.
- DirectX SDK : **có thể tải tại đây.**

- Install DirectX SDK.
Sau khi install xong thì chúng ta sẽ có các thư mục nằm bên trong *C:\Program Files\Microsoft DirectX SDK* như sau



Bin : thư mục Bin chứa các ứng dụng tiện ích hỗ trợ phát triển ứng dụng DX9. Đây là các tool tuyệt vời để ta tùy ý sử dụng.

• DXCapsViewer.exe : Cho chúng ta thấy khả năng support DirectX của hardware như thế nào.
• DXErr.exe : Cho phép chúng ta nhập error code mà DX API trả về để có thể xem được miêu tả về lỗi sai đó
• DXTex.exe : Cho phép chúng ta import bitmaps dùng làm texture, và convert nó thành định dạng texture chuẩn dùng trong DirectX (DDS)
• vsa.exe : Cho phép chúng ta compile vertex shader
• pas.exe : Cho phép chúng ta compile pixel shader

Doc : folder này chứa tài liệu giai đoạn phát triển DirectX. Nó chứa hàng trăm thông tin tham khảo về DirectX. Rất nhiều các function, interface, structure và macro trong DirectX được giải thích trong đây

Include : folder này chứa toàn bộ các C++ header file mà ta cần include vào project để tạo ứng dụng DirectX.

Libs : folder này chứa library filde mà ta cần link vào project để có thể dùng hàm và interface của DirectX.

Redist : folder này chứa các DirectX runtime mà ta có thể ship cho người dùng ứng dụng. Việc thực thi trong folder này cho phép tự động kiểm tra và install DirectX9 runtime vào hệ thống của người dùng. Chúng ta sẽ có các ví dụ về cách sử dụng

Samples : folder này chứa các chương trình mẫu (với source code) thể hiện cách dùng DirectX và các đặc điểm của nó.

SDKDev : thư mục này chứa runtime install application mà nó tự động install với SDK.

* Chú ý là trong môt vài phiên bản khác nhau của DirectX thì các thư mục có thể đặt ở các vị trí khác nhau, ví dụ Bin có thể đặt trong thư mục Utilities
* Chúng ta có thể xem các Sample của DirectX bằng chương trình **DirectX Sample Browser** (vào Start->Program->Microsoft DirectX SDK->DirectX Sample Browser) nơi đây chứa rất nhiều các ví dụ bổ ích về lập trình DirectX

Pasted from <http://gameuit.com/gameuit/showthread.php?287-GameUIT-1st-Challenge-Installing-the-DirectX9-SDK>

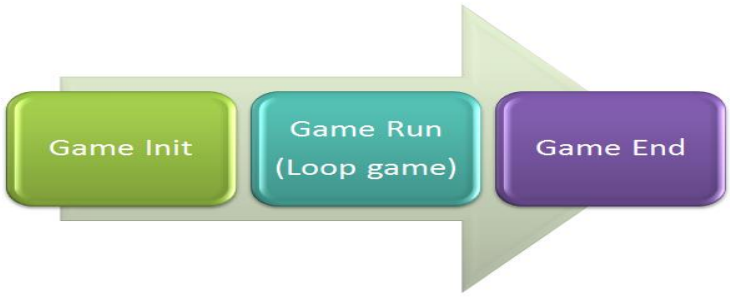
III. Tạo một project với DirectX

1. Mở visual studio.
2. Tạo project mới bằng cách chọn File -> New -> Project. Sau đó chọn ngôn ngữ là visual C++ và chọn empty project.
3. Để có thể sử dụng các API của DirectX thì ta cần chọn đường dẫn đến thư viện DX.

+ Đối với Visual 2008 trở xuống : Tool -> Options -> Projects & solutions -> VC++ Directories.
Phần Show directories for -> chọn Include files -> add đường dẫn của folder include của DX và thường là :
C:\Program Files\Microsoft DirectX SDK (June 2010)\Include
Tương tự chọn library files và add đường dẫn :
C:\Program Files\Microsoft DirectX SDK (June 2010)\Lib\x86
+ Đối với visual studio 2010 bạn phải làm công đoạn trên đối với mỗi project bạn tạo ra. Projects -> " Project name " properties -> VC++ directories. Ở đây bạn sẽ thấy 2 đường dẫn là include directories và library directories bạn chỉ việc add đường dẫn như ở trên.
+ Tùy theo từng máy các bạn cài DirectX ở đâu thì các bạn add đường dẫn tới đó chứ không nhất thiết là đường dẫn ở trên (đó chỉ là mặc định cho Win32 bit).

IV. Cấu trúc của một game

Một game cơ bản dù là 2D hay 3D luôn có 1 vòng đời của game.



- **Game Init** : Là phần khởi tạo cần thiết cho một game như đăng ký (khởi tạo) cửa sổ window, khởi tạo device, input (mouse & keyboard), load các hình ảnh, âm thanh, load model ...

- **Game Run** : Đây là vòng lặp của game nơi xử lý âm thanh, hình ảnh, input, gameplay và nhiều thứ khác nữa.
- **Game End** : Nơi giải phóng các tài nguyên mà đã được cấp phát ở Game Init.

* Trong quá trình lập trình game chúng ta sẽ sử dụng DirectX 9 gồm có : Direct3D, DirectSound, DirectInput, ...
Như vậy bạn đã hoàn thành các bước chuẩn bị cơ bản để bước vào thế giới lập trình game (gian khổ).

Pasted from <http://gameuit.com/gameuit/showthread.php?286-GameUIT-1st-Challenge-Beginning-game-with-DirectX>

Phần 2: Create window with DirectX

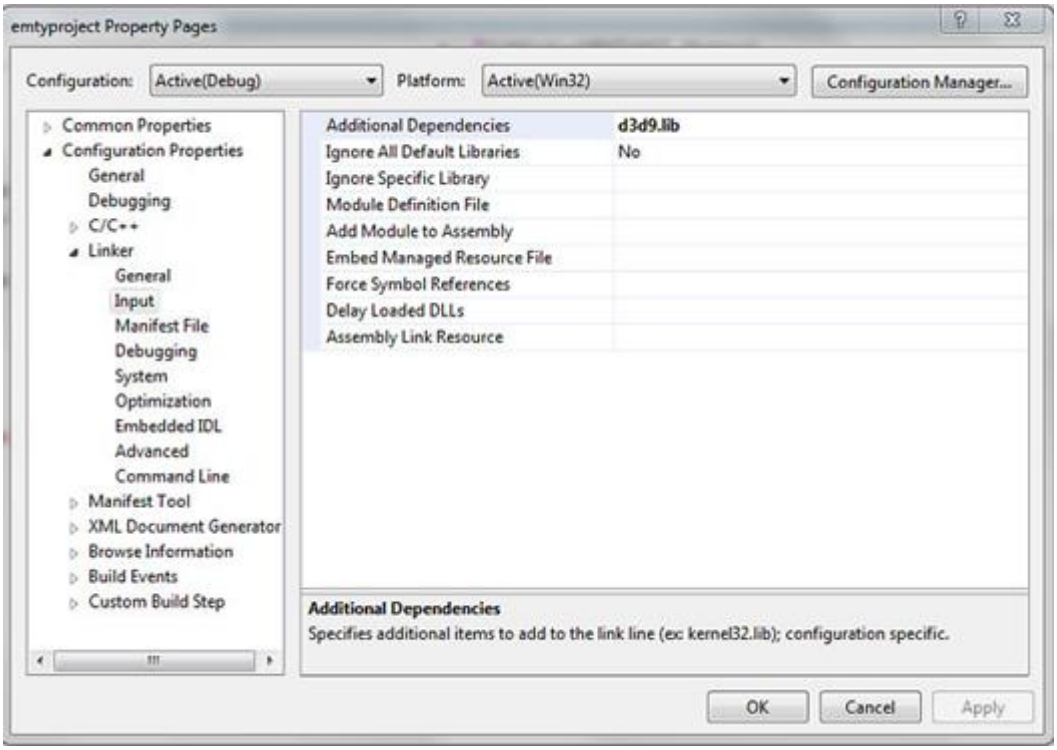
Hãy chắc chắn rằng bạn đã cài đặt bộ DirectX SDK và tạo đường dẫn cho các folder Include và Library.

I. Add thư viện cho project

- Sau khi tạo một empty project, tạo một file .cpp.
- Ta cần include của directX như sau.

```
#include "d3d9.h"
/*Đây là bộ thư viện chứa các phương thức cơ bản DirectX bao g
ồm khởi tạo device, input, window...*/
```

- Sau đó để add thư viện vào ta click chuột phải lên tên project -> chọn propoties.
Tại phần General -> Character Set -> Chọn Use multi-byte character set. (Nếu không làm công đoạn này sẽ gặp lỗi khi sử dụng chuỗi trong chương trình, hoặc bạn có thể sử dụng L"string" mà không cần tùy chọn này).
- Tại phần linker -> input -> Additional dependencies ta nhập vào d3d9.lib.



II. Một số khái niệm cần biết

- HWND : là handle của một window trong một ứng dụng. Một ứng dụng có thể có nhiều window, mỗi window sẽ được đại diện bởi một HWND.

- HINSTANCE : là handle của ứng dụng khi ứng dụng đó được hệ điều hành nạp vào để chạy. Ví dụ chúng ta chạy chương trình netscape.exe 2 lần đồng thời (bằng cách double click trên lên netscape icon 2 lần. Khi đó chúng ta thấy có 2 thể hiện của cùng mộtchương trình netscape, mỗi thể hiện sẽ có một HINSTANCE.

- Hàm CALLBACK : là một hàm sẽ tự động gọi khi có một message được gửi đến. Tương tự như delegate trong C# hoặc con trỏ hàm.

III. Create window

- 3 bước để tạo một cửa sổ là : tạo (đăng ký) định danh cho window, khởi tạo các thuộc tính của window và vẽ nó ra màn hình.

1. Tạo định danh cho window

- Trước tiên bạn cần tạo một HWND cho project của bạn, có thể là global var hoặc member trong class.

```
class CGame
{
```

```
private:
.
.
.
    HWND    _hWnd;
.
.
.
}
```

- Ở đây ta sử dung struct WNDCLASS. Các bạn có thể xem đầy đủ các member của struct này ở đây tuy nhiên ta chỉ quan tâm đến một số điểm quan trọng.

```
WNDCLASS    wndc;
ZeroMemory(&wndc, sizeof(wndc));    // Xem kĩ hơn trên msnd. D
ùng để khởi tạo các giá trị ban đầu.

wndc.hbrBackground    = (HBRUSH)GetStockObject(WHITE_BRUSH);
wndc.hCursor        = LoadCursor(NULL,IDC_ARROW);
wndc.hIcon        = NULL;
wndc.hInstance        = _hInstance;
wndc.lpfnWndProc    = (WNDPROC)_WndProc;
wndc.lpszClassName    = "CGame";

if(!RegisterClass(&wndc))        // Kiểm tra việc đăng ký có t
hành công ?
{
    MessageBox(0,"Can not Register Class","Error",MB_OK);
    return false;
}
```

* Lưu ý : wndc.lpfnWndProc ở đây ta gán tên một hàm (_WndProc – sẽ khởi tạo sau), được ép kiểu. Ở đây nó là hàm xử lý các sự kiện (nằm trong Message Queue) ta muốn xử lý.

2. Khởi tạo thuộc tính cho window

- Để khởi tạo các thuộc tính cho window ta sử dụng hàm CreateWindow

```
HWND WINAPI CreateWindow(
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName,    // Tên trên thanh caption
    DWORD dwStyle,        // Style    xem thêm tại đây : [URL=
"http://msdn.microsoft.com/en-
us/library/ms632600(v=VS.85).aspx"]click here[/URL]
    int x,                    // Tọa độ trên screen
    int y,
    int nWidth,                // Chiều rộng của cửa s
ổ
    int nHeight,
```

```
    HWND hWndParent,         // NULL Game thường chỉ có 1 cửa s
ổ
    HMENU hMenu,                 // NULL Game thường không có m
enu
    HINSTANCE hInstance,     // hInstance của hàm main
    LPVOID lpParam                 // NULL
);
```

Hoặc có thể xem đầy đủ ở MSDN : click here
*** Lưu ý : Tên của lpClassName phải trùng với tên của wndc.lpszClassName. Đây là lỗi mà các bạn hay mắc phải**

3. Thể hiện window ra màn hình

- Bước này đơn giản chỉ cần gọi hàm ShowWindow và UpdateWindow.

```
ShowWindow(_hWnd,SW_SHOWNORMAL);
UpdateWindow(_hWnd);
Dưới đây là một example cho hàm khởi tạo window.
PHP Code:
bool CGame::_InitWindow()
{

    WNDCLASS    wndc;
    ZeroMemory(&wndc, sizeof(wndc));

    wndc.hbrBackground    = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndc.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wndc.hIcon            = NULL;
    wndc.hInstance        = _hInstance;
    wndc.lpfnWndProc    = (WNDPROC)_WndProc;
    wndc.lpszClassName    = "CGame";
    wndc.lpszMenuName    = NULL;
    wndc.style            = NULL;

    if (!RegisterClass(&wndc))
    {
        MessageBox(0, "Can not Register Class", "Error", MB_OK);
        return false;
    }

    _hWnd    = CreateWindow(
        "CGame",
        "Begin 3D",
        WS_OVERLAPPEDWINDOW,
        100,
        100,
        _scrWidth,
        _scrHeight,
        NULL,
        NULL,
        _hInstance,
        NULL);
```

```
    if (!_hWnd)
    {
        MessageBox(0, "Can not Create Window", "Error", MB_OK);
        return false;
    }

    ShowWindow(_hWnd,SW_SHOWNORMAL);
    UpdateWindow(_hWnd);

    return true;
}
```

Pasted from <http://gameuit.com/gameuit/showthread.php?291-GameUIT-1st-Challenge-Create-window-amp-device-with-DirectX>

Phần 3: Tạo Direct3D Object and Direct3D Device

I. Khởi tạo đối tượng Direct3D

- Để khởi tạo device trong DirectX bạn cần tạo 2 đối tượng là LPDIRECT3D9 và LPDIRECT3DDEVICE9. Có thể là global var hoặc member trong class.

```
class CGame
{
private:
...
    LPDIRECT3D9             _d3d;
    LPDIRECT3DDEVICE9         _d3ddv;
...
}
```

*Lưu ý : LP ở đây là long pointer, vì vậy chúng ta cũng có thể khởi tạo như sau :
`IDIRECT3D9 *_d3d;`
- Để khởi tạo cho LPDIRECT3D9 ta sử dụng hàm sau :
PHP Code:
```
_d3d    = Direct3DCreate9(D3D_SDK_VERSION);
```
Các bạn cần phải học thuộc đoạn mã này.
- `LPDIRECT3D9` có 2 nhiệm vụ chính :
+ Device enumeration : việc này có nghĩa là ta dùng nó để kiểm tra khả năng, định dạng, các thông số của card màn hình để có thể tùy chỉnh thích hợp đối với ứng dụng game của ta. (công việc này rất cần thiết vì ngay sau bước này ta sẽ tiến tới bước tạo Device, lúc đó tất cả các thông tin này phải đc xem xét và khai báo).
+ Tạo Device, tức là con trỏ tới `IDIRECT3DDEVICE9` .

Pasted from <http://gameuit.com/gameuit/showthread.php?291-GameUIT-1st-Challenge-Create-window-amp-device-with-DirectX>

II. Tạo Direct3D Device

- Để khởi tạo thuộc tính cho device ta sử dụng struct `D3DPRESENT_PARAMETERS` : xem đầy đủ tại đây.

```
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));

    d3dpp.Windowed         = true;
    d3dpp.BackBufferCount    = 1;
    d3dpp.BackBufferFormat    = D3DFMT_UNKNOWN;
    d3dpp.BackBufferHeight    = 600;
    d3dpp.BackBufferWidth    = 800;
    d3dpp.hDeviceWindow    = _hWnd;
    d3dpp.SwapEffect        = D3DSWAPEFFECT_DISCARD;
```

- `Windowed` : ở windowed mode nếu true và fullscreen nếu false.
- `BackBufferCount` : Số lượng Back buffer sẽ dùng (thường là 1, sẽ nói sau vì khá phức tạp).
- `BackBufferFormat` : Định dạng cho Back buffer. Khi gán `D3DFMT_UNKNOWN` nghĩa là DirectX sẽ tự tạo một format phù hợp cho device.
- `BackBufferHeight`, `BackBufferWidth` : chiều rộng và dài của backbuffer.
- `SwapEffect` : Phương thức swap giữa Front buffer và Back Buffer.

* Front buffer : là một bộ đệm dùng để hiển thị trên màn hình, có thể hiểu nôm na là một ma trận 2 chiều, mỗi phần tử là một điểm ảnh chứa các giá trị của màu sắc, vị trí,...
* Back buffer : nếu chương trình vẽ trực tiếp lên front buffer thì sẽ có hiện tượng giống như bị xé hình, để khắc phục điều này chương trình sẽ vẽ toàn bộ lên Back buffer, sau đó mới copy qua front buffer D3DSWAPEFFECT_COPY hoặc D3DSWAPEFFECT_FLIP (Sẽ nói sau).

Pasted from <http://gameuit.com/gameuit/showthread.php?291-GameUIT-1st-Challenge-Create-window-amp-device-with-DirectX>

III. Khởi tạo Device

- Vì không phải tất cả các máy tính ngày nay đều có card đồ họa rời và hỗ trợ HAL (Hardware Abstraction Layer : click here) nên khi khởi tạo device ta sẽ kiểm tra và tạo device phù hợp.
- Trước tiên ta xem qua cấu trúc hàm CreateDevice.

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

```
);
```

Xem MSDN : click here
- Adapter : là nói tới card đồ họa, do ta chỉ sử dụng card đồ họa chính (primary card) cho nên giá trị ở đây là D3DADAPTER_DEFAULT hoặc để số 0.

- D3DDEVTYPE : Có 4 lựa chọn, chủ yếu nằm trong enum _D3DDEVTYPE là D3DDEVTYPE_HAL = 1, D3DDEVTYPE_REF = 2, D3DDEVTYPE_SW = 3, D3DDEVTYPE_FORCE_DWORD = 0xffffffff.

+ D3DDEVTYPE_HAL : Sử dụng HAL, tức là các thư viện DirectX sẽ được hỗ trợ bởi phần cứng (coi phần Hardware Abstraction Layer). Chú ý là nếu không có xử lý 3D thì không cần HAL hoặc việc thiết lập HAL là không cần thiết. Nếu HAL được tạo thành công thì tức là ứng dụng phải có 3D gì đó. Có nghĩa là quá trình rasteriazation chắc chắn sẽ được làm bởi video card hay cả quá trình transformation và lighting ... chúng ta phải check coi card màn hình có hỗ trợ HAL hay không trước khi yêu cầu Device sử dụng HAL.

+ D3DDEVICETYPE_REF : Nếu HAL không chọn thì HEL là lực chọn của chúng ta (Coi phần Hardware Abstraction Layer) và tùy chọn REF này có nghĩa là ta sẽ sử dụng HEL cho device. Tùy chọn HEL thường được sử dụng để kiểm tra xem hardware có làm việc chính xác hay không. Ví dụ video card maker có thể kiểm tra quá trình phát triển xem việc reference rasterizer của card video xem nó render polygon sáng hay tối như thế nào. Chúng ta nên có phần kiểm tra nếu card không có HAL thì ta phải chỉnh tùy chọn HEL cho device.

+ D3DDEVICETYPE_SW : Tùy chọn này cho ứng dụng của ta chạy trên máy không có card màn hình (hay là card màn hình không hỗ trợ cả HEL). Tất cả sẽ chạy đúng nghĩa trên Cpu.

Dưới đây là một cách đơn giản nhất để khởi tạo device phù hợp (cách này đơn giản nhất, sau này khi học 3D sẽ cần những cách khác vì phải kiểm tra một số thuộc tính riêng).

```
bool CGame::_InitDirectX()
{
    HRESULT hr;

    _d3d    = Direct3DCreate9(D3D_SDK_VERSION);

    if (!_d3d )
    {
        MessageBox(0, "Can not create directX 9", "Error", MB_OK);
        return false;
    }
```

```
    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));

    d3dpp.Windowed               = _windowMode;
    d3dpp.BackBufferCount     = 1;
    d3dpp.BackBufferFormat     = D3DFMT_UNKNOWN;
    d3dpp.BackBufferHeight     = 600;
    d3dpp.BackBufferWidth     = 800;
    d3dpp.hDeviceWindow          = _hWnd;
    d3dpp.SwapEffect          = D3DSWAPEFFECT_DISCARD;

    if(FAILED(_d3d-
>CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, _hWnd,  D3DCREATE_
HARDWARE_VERTEXPROCESSING,  &d3dpp, &_d3ddv)))
            if(FAILED(_d3d-
>CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_REF, _hWnd, D3DCREATE_H
ARDWARE_VERTEXPROCESSING, &d3dpp, &_d3ddv)))
                    if(FAILED(_d3d-
>CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_REF, _hWnd, D3DCREATE_S
OFTWARE_VERTEXPROCESSING, &d3dpp, &_d3ddv)))
                        {
                            MessageBox(0, "Can not create device","E
rror", MB_OK);
                            return false;
                        }

    return true;
}
```

Release device

- Sau khi kết thúc vòng lặp game ta cần giải phóng device. Đơn giản chỉ gọi hàm Release của mỗi đối tượng và set chúng về NULL.

```
void CGame::_Release()
{
    if (_d3d != NULL)
    {
        _d3d->Release();
        _d3d = NULL;
    }
    if (_d3ddv != NULL)
    {
        _d3ddv->Release();
        _d3ddv = NULL;
    }
}
```

Pasted from <http://gameuit.com/gameuit/showthread.php?291-GameUIT-1st-Challenge-Create-window-amp-device-with-DirectX>

IV. Game Loop

- Điều quan trong nhất của một game chính là vòng lặp game, đây là nơi xử lý message, gameplay, input, sound,

graphic, ...
- Các bạn xem qua đoạn code dưới đây rồi mình sẽ giải thích.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstanc
e, LPSTR lpCmdLine, int nShowCmd)
{
    MSG msg;

    CGame    Run(hInstance, SCREEN_WIDTH, SCREEN_HEIGHT, true,
 FRAME_RATE);
    Run._Init();

    ZeroMemory(&msg, sizeof(msg));

    while (1)
    {
        if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
        {
            if (msg.message == WM_QUIT)
                break;

            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        else
        {
            //Run._Run();  // Đoạn này có không cũng được, vì
hiện tại chưa cần nói đến.
        }
    }
    Run._Release();
    return 0;
}

LRESULT CALLBACK CGame::_WndProc(HWND hWnd, UINT message, WPAR
AM wParam, LPARAM lParam)
{
    switch(message)
    {
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
}
```

- Ở đây ta có một vòng lặp while xử lý game, vòng lặp chỉ dừng lại khi nhận được tín hiệu thoát game.
- **PeekMessage** : hàm này lấy message tại Message Queue và truyền vào biến msg. Argument cuối cùng là PM_REMOVE, nghĩa là sau khi lấy được thông điệp, nó sẽ xóa thông điệp ra khỏi hàng đợi (ngoại trừ WM_PAINT). Nếu có message trong window hàm sẽ trả về true, ngược lại else.

* Lưu ý : Khi chạy chương trình bạn sẽ thấy ứng dụng của bạn chiếm CPU lên đến 50%, đừng lo lắng về điều này, nó chỉ lấy CPU idle mà thôi, sở dĩ nó chiếm CPU cao như vậy vì nó liên tục kiểm tra message queue. Một điều quan trọng nữa là trong game không như một application window, ta sử dụng hàm PeekMessage chứ không dùng GetMessage. Điểm khác biệt giữa 2 hàm này là khi có 2, 3 hoặc nhiều messsage trong queue, GetMessage sẽ lấy message và xử lý message đó, đến khi xử lý xong mới lấy message tiếp theo. PeekMessage sau khi lấy message, sẽ xóa ngay message đó, vì vậy trong game ta phải dùng PeekMessage.

- TranslateMessage : xử lý input nhập vào và chuyển sang dạng message và được chuyển vào message queue cho lần xử lý sau.
- DispatchMessage : gửi message đến window proceduce.

- Bây giờ cùng nhìn qua một chút hàm _WndProc : đơn giản chỉ là xử lý message, Ví dụ ta bắt sự kiện WM_DESTROY (sự kiện này có được khi ta nhấn nút close, tức là dấu "X" trên thanh caption). Hàm **DefWindowProc** là hàm window proceduce mặc định của window.

Các bạn có thể tải project example dưới đây để có thể tìm hiểu rõ hơn, cố gắng code lại, mới đâu có thể vừa nhìn vừa code, sau đó cố gắng nhớ các hàm này, điều này rất quan trọng mặc dù sau này có thể ta không còn đụng đến nó nữa.

Code Example : http://www.mediafire.com/download.php?pqak8zufzauo7rh

Một số kinh nghiệm khi học DirectX:
- Các bạn nên tìm hiểu các chữ viết tắt của DirectX như vậy sẽ dễ nhớ hơn, tất nhiên là ý nghĩa nữa và cũng nâng cao trình độ anh văn. Ví dụ WM_DESTROY thì WM là window message, WndProc là window proceduce, ...
- Nên tìm hiểu trên MSDN, ở đây cập nhật đầy đủ và chính xác nhất các hàm, khái niệm.

Phần 4: Surface – Texture

Draw an image with DX : Surface & Texture

Làm thế nào để vẽ một hình lên màn hình bằng DX ?
Có 2 cách để vẽ một hình là dùng surface và texture. Hiện tại rất nhiều người sử dụng texture là chính vì nó có nhiều chức năng hơn surface, tuy nhiên mình cũng đã thăm dò trên google thì texture được xây dựng dựa trên surface và có nhiều function hơn, do đó sử dụng surface nhẹ và nhanh hơn texture.
Surface chỉ nên dùng cho một map scroller game (tức là hình có kích thước khá lớn), ví dụ 1 game có map 5000x600 chẳng hạn, còn các hình nhỏ thì ta nên dùng texture.
Hôm nay mình sẽ hướng dẫn sử dụng surface để load và vẽ một hình, làm nó chuyển động.

I. Một số khái niệm

- Front buffer : bộ đệm chính là những vùng hiển thị trên màn hình có độ phân giải bằng size của window ta tạo ra.
- Back buffer : bộ đệm phụ là vùng đệm dùng để vẽ lên đây, sau khi vẽ hoàn tất sẽ được thể hiện ra front buffer.
• Nguyên nhân : có thể hiểu nôm na nếu bạn vẽ trực tiếp ra front buffer thì hình ảnh hiển thị sẽ rời rạc hay còn gọi là xé hình, cụ thể nếu ta có 100 objects cần được vẽ trên một màn hình đen mất 1s chẳng hạn, màn hình có chu kì là 0.5s vẽ 1 lần hay 2Hz. Như vậy ở giây thứ 0.5 ta sẽ thấy có 50 objects trên màn hình, và giây thứ 1.0 ta sẽ thấy đầy đủ 100 objects, trong khi nếu sử dụng back buffer thì sau khi vẽ đủ 100 object, hàm Present sẽ thể hiện ra màn hình chính xác 100 objects. Ví dụ chỉ mang tính chất minh họa (màn hình máy tính thường có >= 60Hz và 1 game thường có FPS = 60).
- Offscreen surfaces : là vùng trên bộ nhớ đồ hoạ hay hệ thống được dùng để lưu trữ những đối tượng hình hoạ mà game cần sử dụng. Một ví đụ điển hình là khi ta cần vẽ 60 hình tròn lên màn hình thì ta không phải load hình ra 60 lần mà chỉ cần lưu 1 lần tại offscreen surface này.

II. Surface

1. Khởi tạo đối tượng

```
LPDIRECT3DSURFACE9          _surface;
LPDIRECT3DSURFACE9          _backbuffer;
```

_surface : dùng để lưu hình ảnh sau khi được load.
_backbuffer : dùng để vẽ hình ảnh lên back buffer.

2. Tạo offscreen và load image

- Để load một hình thì ta cần tạo offscreen surface cho tấm hình đó thông qua hàm CreateOffscreenPlainSurface : MSDN

```
HRESULT CreateOffscreenPlainSurface(
[in]            UINT Width,
[in]            UINT Height,
[in]            D3DFORMAT Format,
[in]            D3DPOOL Pool,
```

```
[out, retval]  IDirect3DSurface9 **ppSurface,
[in]           HANDLE *pSharedHandle
);
```

- Width, height : là size của hình cần load.
- Format : kiểu format của surface, xêm thêm tại đây , thường ta sẽ dùng D3DFMT_X8R8G8B8.
- Pool : nơi lưu trữ tấm hình, thường để D3DPOOL_DEFAULT.
- **ppsurface : con trỏ đến surface.
- *pSharedHandle : ta để là null.

Để load một hình ảnh ta sử dụng hàm D3DXLoadSurfaceFromFile : MSDN

```
HRESULT D3DXLoadSurfaceFromFile(
  __in     LPDIRECT3DSURFACE9 pDestSurface,
  __in     const PALETTEENTRY *pDestPalette,
  __in     const RECT *pDestRect,
  __in     LPCTSTR pSrcFile, // file path
  __in     const RECT *pSrcRect,
  __in     DWORD Filter,
  __in     D3DCOLOR ColorKey,
  __inout  D3DXIMAGE_INFO *pSrcInfo
);
```

- pDestSurface : surface lưu hình ảnh.
- Filter : Cái này chưa quan tâm.
- ColorKey : là màu ta muốn transparent, ví dụ ta có 1 tấm hình có 1 nữa màu vàng, nữa còn lại màu đỏ. Ta chỉ định colorkey là màu đỏ thì khi render ta sẽ chỉ thấy phần màu vàng, phần màu đỏ sẽ trong suốt.
- *pSrcRect : sử dụng khi muốn load 1 phần của tấm ảnh. (RECT là một kiểu struct gồm có top, bottom, left, right dùng để xác định một vùng rectangle nào đó). Set null để load toàn bộ tấm hình.
- *pDestRect : chỉ định vùng rectangle trong surface. Set null chỉ định toàn bộ surface.

Dưới đây là một ví dụ về *psrcRect và *pDestRect :

Ví dụ cho dễ hiểu :

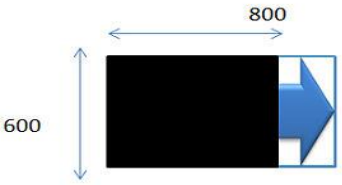
Resolution : 800x600

```
RECT pSrcRect;
pSrcRect.Top = 0;
pSrcRect.Bottom = 600;
pSrcRect.Left = 400;
pSrcRect.Right = 800;
```

Ta sẽ có :

```
RECT pDestRect;
pDestRect.Top = 0;
pDestRect.Bottom = 600;
pDestRect.Left = 0;
pDestRect.Right = 800;
```

Ta sẽ có :



Như ta thấy hình đã bị scale lại, phần trống khi render ra sẽ có màu đen

3. Getbuffer cho việc vẽ.

Để có thể vẽ lên back buffer ta cần chỉ định con trỏ trỏ đến back buffer thông qua hàm GetBackBuffer: MSDN trỏ đến backbuffer để chuẩn bị vẽ

```
HRESULT GetBackBuffer(
  [in]            UINT  iSwapChain,
  [in]            UINT BackBuffer,
  [in]            D3DBACKBUFFER_TYPE Type,
  [out, retval]   IDirect3DSurface9 **ppBackBuffer
);
```

- iSwapChain : MSDN nói không rõ lắm, thường để là 0.
- Backbuffer : chỉ định số lượng BackBuffer cần dùng, BB = 0 nghĩa là sẽ có 1 backbuffer.
- Type : luôn luôn điền D3DBACKBUFFER_TYPE_MONO.
- **ppBackBuffer : con trỏ đến BackBuffer

Ví dụ cho hàm LoadSurface:

```
LPDIRECT3DSURFACE9 CGame::_LoadSurface(LPCSTR filepath)
{
    LPDIRECT3DSURFACE9 surface = NULL; // lưu trữ một image

    HRESULT result;

    D3DXIMAGE_INFO _info;
    D3DXGetImageInfoFromFile(filepath,&_info);
```

```
    result = _d3ddv-
>CreateOffscreenPlainSurface(_info.Width,_info.Height,D3DFMT_X8R8G8B
8,D3DPOOL_DEFAULT,&surface,NULL);

    if(result != D3D_OK)
        return 0;

    result = D3DXLoadSurfaceFromFile(surface,NULL,NULL,filepath,NULL
,D3DX_DEFAULT,0,NULL);

    _d3ddv-
>GetBackBuffer(0,0,D3DBACKBUFFER_TYPE_MONO,&_backbuffer);

    return surface;
}
```

Sử dụng như sau :
PHP Code:
```
_surface = _LoadSurface("Rectangle.png");
```

4. Render

Để render một hình ta sử dụng hàm StretchRect : MSDN

```
HRESULT StretchRect(
  [in]  IDirect3DSurface9 *pSourceSurface,
  [in]  const RECT *pSourceRect,
  [in]  IDirect3DSurface9 *pDestSurface,
  [in]  const RECT *pDestRect,
  [in]  D3DTEXTUREFILTERTYPE Filter
);
```

- *pSourceSurface : surface của hình ảnh.
- *pSourceRect : xác định phần nào sẽ vẽ của hình, null là sẽ vẽ toàn bộ.
- *pDestSurface : backbuffer.
- *pDestRect : cái này quan trọng, dùng để xác định sẽ vẽ hình lên phần nào trên màn hình, nếu để null, hình sẽ được scale toàn màn hình. Vì vậy cần xác định cẩn thận để vẽ cho chính xác, ở đây nếu bạn muốn scale hình to ra 2 lần chỉ việc chỉ định rect này to ra là được.
- Filter : Xem sơ ở đây để hiểu thêm đôi chút . Hàm này chỉ chấp nhận 3 kiểu là `D3DTEXF_NONE, D3DTEXF_POINT, D3DTEXF_LINEAR và D3DTEXF_NONE.` Nếu render đúng kích cỡ thì không cần filter.

Ví dụ của hàm render :

```
void CGame::_Run()
{
    // Clear screen before draw
    _d3ddv-
>Clear(0, 0, D3DCLEAR_TARGET, 0x0af0f0c3, 1.0f, 0);
```

```
    // Update position of rectangle
    Rec.Update();

    // Check if rectangle collide with screen
    if (Rec.X <= 0 || (Rec.X + Rec.Width) >= SCREEN_WIDTH)
        Rec.VecX *= -1;
    if (Rec.Y <= 0 || (Rec.Y + Rec.Height) >= SCREEN_HEIGHT)
        Rec.VecY *= -1;

    // Specify position of rectangle on screen
    RECT des;
    des.top = Rec.Y;
    des.bottom = des.top + Rec.Height;
    des.left = Rec.X;
    des.right = des.left + Rec.Width;

    // Draw
    if(_d3ddv->BeginScene())
    {
        _d3ddv-
>StretchRect(_surface, NULL, _backbuffer, &des, D3DTEXF_NONE);

        _d3ddv->EndScene();
    }

    // Render to front buffer
    _d3ddv->Present(0, 0, 0, 0);
}
```

- Hàm Clear : dùng để xóa toàn bộ màn hình về một màu nhất định, phải có hàm này nếu không khi vẽ nhiều hình sẽ bị chồng lên nhau. Ở đây quan tâm đối số thứ 3 chính là màu sẽ clear. Bạn có thể tìm hiểu kĩ hơn trên msdn.
- Hàm BeginScence : Báo với chương trình ta sẽ render hình ảnh, nếu hàm này trả về false tức là ta đã bị lost device hoặc một số lỗi khác.

5. Tổng kết

- Support file types : .bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm, and .tga.
- Ưu điểm : nhẹ, nhanh.
- Khuyết điểm : chỉ có scale, không có rotation, flip... Chỉ cần một phần nhỏ của hình vượt quá giới hạn màn hình, hình sẽ không render nữa. ví dụ Rect.Right > SCREEN.Width chẳng hạn. Một điểm yếu nữa là không có kênh alpha.

Đây là project example render một hình vuông di chuyển và va chạm với các thành : Click here

II. Texture

DirectX tạo ra cho chúng ta giao diện **IDirect3DTexture9**, nó cung cấp các phương thức để ta có thể thao tác trên Texture resource.
Texture resources là cái giống j nhờ nó là một tập hợp cấu trúc của dữ liệu được thiết kế để lưu trữ các texel. Ôi càng nói càng rối.. texel là gì? Texel nghe quen quen :-?. Chắc hẳn ai cũng đã biết Picture và Pixel rồi hen. Đơn giản cái này cũng như vậy Texture-Texel. Nó chỉ khác nhau về cách thức làm việc và lưu trữ thôi. DirectX sẽ sử dụng texture filtering để điều kiển việc lấy mẫu và nội suy texel thành pixel rồi cho lên màn hình..
http://msdn.microsoft.com/en-us/libr...ture_Resources

1. Load Texture

Quay lại vấn đề làm sao để tạo ra một Texture từ 1 file ảnh? (vd : image.png).
Đầu tiên ta có 2 kiểu
(type) **LPDIRECT3DTEXTURE9** và **PDIRECT3DTEXTURE9** là con trỏ tới IDirect3DTexture9 interface.
Để tạo một pointer (con trỏ) tới một IDirect3DTexture9, ta gọi phương thức **IDirect3DDevice9::CreateTexture** hay bất cứ cái nào ở dưới cũng được

- **D3DXCreateTexture**
- **D3DXCreateTextureFromFile**
- **D3DXCreateTextureFromFileEx**
- **D3DXCreateTextureFromFileInMemory**
- **D3DXCreateTextureFromFileInMemoryEx**
- **D3DXCreateTextureFromResource**
- **D3DXCreateTextureFromResourceEx**

Hàm D3DXCreateTextureFromFileEx:

```
HRESULT D3DXCreateTextureFromFileEx(
__in LPDIRECT3DDEVICE9 pDevice,
__in LPCTSTR pSrcFile,
__in UINT Width,
__in UINT Height,
__in UINT MipLevels,
__in DWORD Usage,
__in D3DFORMAT Format,
__in D3DPOOL Pool,
__in DWORD Filter,
__in DWORD MipFilter,
__in D3DCOLOR ColorKey,
__inout D3DXIMAGE_INFO *pSrcInfo,
__out PALETTEENTRY *pPalette,
```

```
__out LPDIRECT3DTEXTURE9 *ppTexture
);
```

D3DXIMAGE_INFO: http://msdn.microsoft.com/en-us/libr...79(VS.85).aspx

OK, bây giờ sẽ là các bước load image.png -> IDirect3DTexture9 image.

```
LPDIRECT3DTEXTURE9 CreateImageFromFile(LPDIRECT3DDEVICE9 d3ddv, LPWSTR FilePath)
{
HRESULT result;
/*Trước tiên ta khai báo biến kiểu LPDIRECT3DTEXTURE9. Để trỏ đến cái IDirect3DText
ure9 sắp được tạo*/
LPDIRECT3DTEXTURE9 _Image;
 /*Tạo D3DIMAGE_INFO*/
D3DXIMAGE_INFO info;
 /*Load image info từ file image.png*/
result = D3DXGetImageInfoFromFile(FilePath,&info)
 if (result!=D3D_OK) return null;
 /*Load texture resource*/

result = D3DXCreateTextureFromFileEx(
        d3ddv,                  // LPDIRECT3DDEVICE9
        FilePath,                  // LPCTSTR
        info.Width,              // UINT Width
        info.Height,              // UINT Height
        1,                      // UINT MipLevels
        D3DUSAGE_DYNAMIC,          // DWORD Usage
        D3DFMT_UNKNOWN,          // D3DFORMAT Format
        D3DPOOL_DEFAULT,          // D3DPOOL Pool
        D3DX_DEFAULT,              // DWORD Filter
        D3DX_DEFAULT,              // DWORD MipFilGetinter
        D3DCOLOR_XRGB(0,0,0),      // D3DCOLOR ColorKey
        &info,                  // D3DXIMAGE_INFO *pSrcInfo
    NULL,                  // PALETTEENTRY *pPalette
        &_Image);                  // LPDIRECT3DTEXTURE9 *ppTexture

    if (result!=D3D_OK) return null;

    return Image;
}
```

2. Draw Texture

Load xong rồi giờ vẽ lên thôi.
Ta sẽ sử dụng giao diện **ID3DXSprite** để vẽ 1 texture.
Giao diện **ID3DXSprite** cung cấp một tập hợp các phương pháp đơn giản hóa quá trình vẽ bằng cách sử dụng Microsoft Direct3D.
ID3DXSprite: http://msdn.microsoft.com/en-us/libr...49(VS.85).aspx

```
/*Khai báo biến kiểu LPD3DXSPRITE*/
LPD3DXSPRITE _SpriteHandler;
/*Tạo LPD3DXSPRITE*/
D3DXCreateSprite(d3ddv,&_SpriteHandler);
/*Vẽ nè*/
void drawTexture(LPDIRECT3DTEXTURE9 image, const RECT *srcRect, D3DXVECTOR3 positio
n, D3DCOLOR Color)
{
    if( NULL == _pd3dDevice )
        return;

    _spriteHandler->Draw(
```

```
        srcTexture,
        srcRect,
        NULL,
        &position,
        Color);
}
```

OK! Gần xong rồi
Giờ vào hàm Render();

```
_pd3dDevice->BeginScene();

    _spriteHandler->Begin(D3DXSPRITE_ALPHABLEND); <------ chú ý
    // Đặt hàm drawTexture tại đây
    _spriteHandler->End(); < ---------------------------- chú ý

    _pd3dDevice->EndScene();
```

Vấn đề:
1.. Làm sao để 1 hình vẽ trước không bị "đè" bởi một hình phía trước?
2.. Làm sao để xoay một texture?
Giải quyết:

3. Depth

Vấn đề 1:

Để ý tham số z trong position ở trên chúng ta gán bằng 0. Ứng với giá trị của z [0, 1] sẽ là từ gần màn hình tới sâu vào phía trong (front to back). Giao diện ID3DXSprite có sẵn cơ chế sắp xếp các texture theo z (depth).

A. Syntax
```
HRESULT Begin([in] DWORD Flags);
```

B. Parameters
Flags [in] DWORD
Combination of zero or more flags that describe sprite rendering options. For this method, the valid flags are: D3DXSPRITE_ALPHABLEND

```
• D3DXSPRITE__BILLBOARD
• D3DXSPRITE_DONOTMODIFY_RENDERSTATE
• D3DXSPRITE_DONOTSAVESTATE
• D3DXSPRITE_OBJECTSPACE
• D3DXSPRITE__SORT_DEPTH_BACKTOFRONT
• D3DXSPRITE__SORT_DEPTH_FRONTTOBACK
• D3DXSPRITE__SORT_TEXTURE
```

4. SetTransform

Vấn đề 2:
Giao diện ID3DXSprite còn cung cấp một cơ chế biến đổi tọa độ của texture trước khi vẽ :

A. Syntax
HRESULT SetTransform([in] const D3DXMATRIX *pTransform);

B. Parameters
pTransform [in] D3DXMATRIX Pointer to a D3DXMATRIX that contains a transform of the sprite from the original world space. Use this transform to scale, rotate, or transform the sprite.

C. Using

```
D3DXVECTOR2 rotate_center;

rotate_center.x = center.x; // tâm của texture so với màn hình
rotate_center.y = center.y; // tâm của texture so với màn hình

D3DXMATRIX t1;

int i;
i = rand()%2;
D3DXMatrixAffineTransformation2D(
    &t1,                          // result
    1.0f,                          // scaling
    &rotate_center,                  // rotation center
    angle,                       // rotation angle
    NULL                           // translation
    );

_spriteHandler->SetTransform(&t1);

_spriteHandler->Draw(…)
```

PS: Nếu muốn tìm hiểu về texture (texture là gì? nó dùng để làm gì?) cách tốt nhất là mở file windows_graphics.chm vào Topics Direct3D 9 > Programming Guide for Direct3D 9 > Direct3D Textures (Direct3D 9) và đọc. Có rất nhiều thứ thú vị đang chờ các bạn sẽ cố gắng làm thêm vài tut về cái texture này nữa.

Nhiệm vụ của các bạn trong tuần này là **tạo ra một lớp Texture**, gồm một số hàm tiện ích như :
- Khởi tạo, có tham số đường dẫn đến file hình luôn.
- Hàm hủy, gọi hàm hủy Texture.
- Hàm vẽ texture với nhiều loại tham số khác nhau. Các bạn nộp bài ngay tại đây nha.

Pasted from <http://gameuit.com/gameuit/showthread.php?373-Training-online-03-09-2011-Texture&highlight=texture>

Phần 5: Sprite – Animation

I. Sprite Class

Animation - Chuyển động trong game !
Để tạo được một chuyển động, trước tiên bạn phải có một bức hình chứa các frame liên tục của chuyển động đó :

Để load hình ảnh vào trong directX, và vẽ hình, chúng ta cần có một cái gì đó chứa nó. Vậy cái gì đó là cái gì. Đó là texture (LPDIRECT3DTEXTURE9). Còn để thực hiện được thao tác vẽ, chúng ta lại cần đến Sprite (LPD3DXSPRITE). Chúng ta sẽ tập hợp chúng lại thành một class có tên là Sprite :

Sprite.h:

```
#ifndef _SPRITE_
#define _SPRITE_

#include "d3d9.h"
#include "d3dx9.h"
class Graphic;
class Animation;    //Cái này là cái class tạo ra chuyển động, lát sẽ nói sau
class Sprite
{
private:
    LPDIRECT3DTEXTURE9 image;  //Con trỏ, chứa địa chỉ của texture được load vào ga
me
    LPD3DXSPRITE spriteHandler;   //Con trỏ chứa Sprite, dùng để vẽ
public:
    int        width;                    //Chiều rộng bức hình
    int        height;                  //Chiều cao bức hình

    Sprite();
    Sprite(Graphic* g,char* path);
    void Draw(int x,int y, float xScaleRatio, float yScaleRatio,D3DXVECTOR2 vRotatt
eCenter,float angle,D3DCOLOR color,Animation *spriteInfo);
};
```

Sprite.cpp :

```
#include "Sprite.h"
#include "Graphic.h"             //Là một class chứa các hàm, biến của directX, như
 device, sprite
#include "Animation.h"
Sprite::Sprite(){};
Sprite::Sprite(Graphic* g,char* path)
{
    spriteHandler=g->spriteDX;         //Lấy sprite để vẽ
    D3DXIMAGE_INFO imageInfo;      //Biến lấy thông tin bức ảnh
    D3DXGetImageInfoFromFile(path,&imageInfo);
    width=imageInfo.Width;
    height=imageInfo.Height;
    D3DXGetImageInfoFromFile(path,&imageInfo);
    D3DXCreateTextureFromFileEx(g->device,
        path,
        imageInfo.Width,
        imageInfo.Height,
        1,                          //Mipmap level
        D3DPOOL_DEFAULT,
        D3DFMT_UNKNOWN,
        D3DPOOL_DEFAULT,
        D3DX_DEFAULT,
        D3DX_DEFAULT,
        D3DCOLOR_XRGB(255,255,255),
        &imageInfo,
        NULL,
        &image);
}
```

```
void Sprite::Draw(int x,int y, float xScaleRatio, float yScaleRatio,D3DXVECTOR2 vRo
tatteCenter,float angle,D3DCOLOR color,Animation *spriteInfo)
{
    spriteHandler->Begin(D3DXSPRITE_ALPHABLEND);

    D3DXMATRIX mTransform;
    D3DXMatrixTransformation2D(&mTransform,NULL,0,&D3DXVECTOR2(xScaleRatio,yScaleRa
tio),&vRotatteCenter,angle,&D3DXVECTOR2((float)x,(float)y));
    spriteHandler->SetTransform(&mTransform);
    spriteHandler->Draw(image,&spriteInfo-
>rect,NULL,&D3DXVECTOR3(0,0,1.0f),color);

    spriteHandler->End();
}
```

Trong Sprite.cpp có phần hàm Draw là có vẻ hơi khó nuốt một xíu. Các tham số truyền vào gồm tọa độ để vẽ x, y, tỷ lệ khi vẽ so với ảnh gốc xScaleRatio, yScaleRatio, trục quay vRotateCenter, góc quay angle, màu color, và một con trỏ trỏ đến đối tượng Animation, để lấy vùng hình chữ nhật trên bức hình được vẽ.

Nãy giờ có nhắc đến Animation, là gì nhỉ ? Một class lưu và xử lý việc chuyển đổi giữa các frame trong hình.

Animation.h

```
#ifndef _ANIMATION_
#define _ANIMATION_

#include "windows.h"
#include "d3d9.h"
#include "d3dx9.h"
#include "Game.h"

class Sprite;
class Animation
{
private:
    float    waitNextImage;                //Chờ chuyển hình
    int        widthOfFrame;                //Chiều rộng
    int        heightOfFrame;                //Chiều cao
    int        numImageInRow;                //Số hình trên một hàng
    int        numImageInColumn;            //Số hình trên một cột

public:

    int        index;                        //Chỉ số hình
    float    timePerImage;                //Thời gian chuyển hình
    RECT    rect;                        //Hình chữ nhật trong hình

    Animation();
    Animation(int _numImageInRow, int _numImageInColumn,float _timePerImage,Sprite*
 sprite);
    ~Animation();
    void Update(int indexFirstImage,int indexLastImage,float TPF);
};
#endif
```

Nhìn vào tên gọi, chắc các bạn cũng hình dung được phần nào. Giờ chúng ta sẽ đi ngay vào **Animation.cpp**

```
#include "Animation.h"
#include "Game.h"
#include "Graphic.h"
#include "Sprite.h"

Animation::Animation()
{}
Animation::Animation(int _numImageInRow,int _numImageInColumn,float _timePerImage,S
prite *sprite)
{
    numImageInRow=_numImageInRow;
    numImageInColumn=_numImageInColumn;
    timePerImage=_timePerImage;
    widthOfFrame=sprite->width/numImageInRow;
    heightOfFrame=sprite->height/numImageInColumn;
    index=0;
    waitNextImage=0;
    rect.left=rect.top=0;
    rect.right=widthOfFrame;
    rect.bottom=heightOfFrame;
}
Animation::~Animation()
{}
void Animation::Update(int indexFirstImage,int indexLastImage,float TPF)
{
    if(indexFirstImage<0||indexLastImage>numImageInColumn*numImageInRow)
        return;
    waitNextImage+=TPF;
    if(waitNextImage>=timePerImage)    //Chờ đến khi quá thời gian quy định thì chu
yển framekế tiếp
    {
        if(index<indexLastImage)        //Tăng chỉ số
            index++;
        else
            index=indexFirstImage;
        waitNextImage=0;

        rect.top=(index/numImageInRow)*heightOfFrame;    //Xác định lại rect -
 cái sẽ được lôi ra trong hàm Draw trên sprite (không tin kéo lên coi)
        rect.left=(index%numImageInRow)*widthOfFrame;
        rect.bottom=rect.top+heightOfFrame;
        rect.right=rect.left+widthOfFrame;
    }
}
```

Mỗi đối tượng cần chuyển động sẽ được add một Animation. Đối tượng nào cần vẽ sẽ được vẽ bằng class Sprite. Đến đây có bạn hỏi sao không gộp Animation và Sprite lại làm một. Lý do là có thể hai đối tượng có cùng hình ảnh (thì sẽ dùng cùng một Sprite), nhưng chưa chắc đã cùng một trạng thái chuyển động (vậy nên Animation sẽ khác nhau).
Trước đây mình đã từng gộp chung lại, nhưng nghe mấy anh nói, mới thấy việc đó là không cần thiết, vì sẽ luôn phải load hình, dù cho các đối tượng sử dụng chung một hình.

Đến đây chắc các bạn đã hiểu sơ qua về cách thức tạo ra chuyển động. Còn chần chờ gì nữa, bắt tay code thử xem sao (đừng copy nha).

Pasted from <http://gameuit.com/gameuit/showthread.php?102-TECH-Animation>

II. Time In Game

Real-time in 2D game

Chào các bạn , tuần trước các bạn đã được học về cách load và vẽ 1 surface/texture lên rồi , tuần này chúng ta sẽ sử dụng texture và sprite sheet để tạo ra những chuyển động của 1 object.

Ngoài ra chúng ta sẽ đi thêm vào các hiệu ứng đối với 1 sprite như scale , rotate , translate ...

Nếu bạn nào có làm thử rồi thì chúng ta thấy rằng , những hình ảnh đó chuyển đổi rất nhanh , đó là do tốc độ xử lý của CPU , CPU càng mạnh thì mỗi giây nó chạy game loop càng cao , vì thế ta phải giới hạn số khung hình lại ở mức thường thấy là 60 frame per second (FPS)

Các bạn xem trước 1 số hàm :

D3DXMatrixTransformation2D

QueryPerformanceFrequency

QueryPerformanceCounter

timeGetTime

Pasted from <http://gameuit.com/gameuit/showthread.php?435-Game-2D-Tu%26%237847%3Bn-4-19-11-2011-Sprite-animation-and-Real-time-in-2D-game>

The Performance Time

Phần này chúng ta sẽ nghiêm cứu khái quát về việc quản lý thời gian trong 1 chương trình game, các hàm thông dụng liên quan tới nó. Đây là một khái niệm rất quan trọng vì nó sẽ ảnh hưởng rất nhiều tới các hoạt động trong game.

Đầu tiên ta sẽ biết rằng các hàm quản lý performance timer nằm trong thư viện window cho nên ta phải include header file <Windows.h>

Chúng ta sẽ quan tâm tới một hàm mà nó trả về 1 đơn vị thời gian trong một thời điểm được gọi là count (một đơn vị để miêu tả thời gian đã trôi qua), `QueryPerformanceCounter(...)` bên trong hàm sẽ có 1 thông số là giá trị mà nó trả về dưới dạng _int64. Ta coi thử cú pháp của nó

```
__int64 prevTimeStamp = 0;
```
QueryPerformanceCounter((LARGE_INTEGER*)&prevTimeStamp);
Để tính toán thời gian làm việc giữa 2 điểm khác nhau, ta đơn giản gọi nó 2 lần ở 2 thời điểm, sau đó trừ ra ta sẽ tính được số cout (thời gian) mà chương trình đã trải qua.

```
__int64 A = 0;
QueryPerformanceCounter((LARGE_INTEGER*)&A);
```

/* Do work */

```
__int64 B = 0;
QueryPerformanceCounter((LARGE_INTEGER*)&B);
```

Theo ví dụ như trên thì thời gian trôi qua là B – A
Tuy nhiên có vẻ không có lợi cho chúng ta khi tính toán với giá trị count (giá trị mà hàm trả về), bởi vì ta cần xử lý giá trị nào đó liên quan tới second, chứ không phải là count, do đó ta phải tìm cách chuyển số count này sang second. May mắn thay chúng ta cũng có được một hàm đáp ứng được nhu cầu đó QueryPerformanceFrequency. Ta hãy coi thử cú pháp của nó

```
__int64 cntsPerSec = 0;
QueryPerformanceFrequency((LARGE_INTEGER*)&cntsPerSec);
```

QueryPerformanceFrequency sẽ trả về số lượng count trong 1 second, từ giá trị đó ta có thể chuyển tính số second trong 1 count.

```
float secsPerCnt = 1.0f / (float)cntsPerSec;
```

Có số second trong 1 count, ta có thể chuyển số count mà ta có từ hàm QueryPerformenceCounter sang số second.
Trên đây là một chút giới thiệu về các hàm ta sử dụng. Chúng ta sẽ đi vào vấn đề quan trọng trong việc sử dụng các hàm này cho chương trình của mình.

Time Differential Between Frames

Tính toán thời gian giữa các Frame rất đơn giản, ta chỉ việc tính ra số count giữa 2 frame rồi từ đó chuyển nó sang đơn vị second. (Lưu ý do việc phải chuyển đổi từ count sang second nên ta phải sử dụng hàm QueryPerformanceFrequency để tính số second trong 1 count). Ta coi thử đoạn code thực thi

```
int D3DApp::run()
{
   MSG  msg;
```

```
    msg.message = WM_NULL;

    __int64 cntsPerSec = 0;
    QueryPerformanceFrequency((LARGE_INTEGER*)&cntsPerSec);
    float secsPerCnt = 1.0f / (float)cntsPerSec;

    __int64 prevTimeStamp = 0;
    QueryPerformanceCounter((LARGE_INTEGER*)&prevTimeStamp);

    while(msg.message != WM_QUIT)
    {
        // If there are Window messages then process them.
        if(PeekMessage( &msg, 0, 0, 0, PM_REMOVE ))
        {
            TranslateMessage( &msg );
            DispatchMessage( &msg );
        }
        // Otherwise, do animation/game stuff.

        else
        {
            if( !isDeviceLost() )
            {
               __int64 currTimeStamp = 0;
               QueryPerformanceCounter((LARGE_INTEGER*)&currTimeStamp);
               float dt = (currTimeStamp - prevTimeStamp)*secsPerCnt;

               updateScene(dt);
               drawScene();

               prevTimeStamp = currTimeStamp;
            }
        }
    }
    return (int)msg.wParam;
}
```

Frame Per Second (FPS) Caculation

Cách tính FPS là ta tính số lượng Frame trong 1s, ta có 1 biến numFrames tính số lượng Frame xảy ra. Như vậy mỗi lần Update ta tăng số Frame này lên, đồng thời ta dùng 1 biến timeElapsed để đếm thời gian đã trôi qua, biến này sẽ được cộng dồn giá trị khoảng thời gian giữa 2 Frame. Cho tới khi timeElapsed vượt qua 1.0f, tức là nó đã qua 1s. ta sẽ xuất giá trị Frame mà ta đếm được ở numFrames. Như vậy ta có số FPS. Lưu ý là khi đã qua 1s, ta khởi gán lại các giá trị timeElapsed và numFrames về 0 để tính FPS tiếp theo.

```
void GfxStats::update(float dt)
{

      static float numFrames   = 0.0f;
      static float timeElapsed = 0.0f;

      // Đếm frames và thời gian trôi qua
      numFrames += 1.0f;
      timeElapsed += dt;

      // Kiểm tra xem đã qua đủ 1s chưa, nếu đủ rồi thì xuất số FPS
      if( timeElapsed >= 1.0f )
      {
            mFPS = numFrames;
```

```
            // Quay lại đếm FPS từ đầu
            timeElapsed = 0.0f;
            numFrames   = 0.0f;
        }
}
```

Bài viết dựa trên tài liệu "Core Techniques and Algorithms in Game Programming"

Pasted from <http://gameuit.com/gameuit/showthread.php?14-Tutorial-The-Performance-Time>

Phần 6: Input

I. Input Object

Trong bài này mình xin giới thiệu về direct input , về cách khai báo , khởi tạo 1 input device , và cách sử dụng nó.

Direct Input cung cấp cho ta khả năng truy xuất đến các thiết bị ngoại vi của PC như chuột , bàn phím , hay là joystick , gamepad...

Để sử dụng được direct input, các hàm của nó, bạn cần phải add thư viện của nó vào : dinput8.lib
Cách add thư viện tham khảo thêm **tại đây**

Sau đó là add file header vào : dinput.h

1. Khởi tạo Input Object và Input Device

Ta khai báo input object và input device như thế này (Nên tạo hẳn 1 class Input để sử dụng sau này)

```
Class Input
{
//some code…..

        LPDIRECTINPUT8              m_pInput ;
        LPDIRECTINPUTDEVICE8        m_pMouseDevice;
        LPDIRECTINPUTDEVICE8        m_pKeyBoardDevice;

//some code……
}
```

Ở đây ta sử dụng 2 device để handle 2 thiết bị là Chuột và Bàn phím, nếu có thêm khác thiết bị khác, thì ta phải khai báo thêm.

Thế là hoàn tất việc khai báo, tiếp theo ta cần làm là khởi tạo Direct Input object bằng cách sử dụng hàm `DirectInput8Create` :

```
bool Input::CreateInput()
{
    HRESULT hr // dùng để kiểm tra việc khởi tạo có thành công hay không
    hr = DirectInput8Create(GetModuleHandle(NULL) , // hInstance của main, ta sử dụng hàm GetModuleHandle trong trường hợp lười phải truyền hInstance từ main vào
                            DIRECTINPUT_VERSION, // always pass this parameter
                            IID_IDirectInput8, // Version của direct input
                            (void**)&m_pInput , // Con trỏ 2 chiều kiểu void trả về giá trị cho Input object
                           NULL // ta không sử dụng tham số này, nên luôn là NULL
                                                );
    If(FAILED(hr))
    {
       MessageBox(0,&#8221;Can't init DX input","Error",MB_OK);
       return false;
    }

    return true;

}
```

Tiếp theo là dùng Input Object vừa tạo ra để khởi tạo các Input device mà ta muốn.Để khởi tạo Input device ta sử dụng hàm sau:

```
m_pInput-
>CreateDevice(GUID_SysKeyBoard, // Loại device mà bạn muốn khởi tạo
                                &m_pKeyBoardDevice, // con trỏ thuộc kiểu Input device lưu thông tin và địa chỉ của thiết bị vừa khởi tạo
                                NULL //Tham số này luôn luôn NULL
                                     );
```

Tham số đầu của hàm CreateDevice có 3 kiểu thường sử dụng :
GUID_SysMouse, GUID_SysKeyBoard và GUID_SysJoystick
Nhưng trên MSDN khuyên không nên truyền giá trị GUID_SysJoystick vào, bạn có thể tìm hiểu thêm trên msdn.

because it is a product GUID, not an instance GUID
OK, vậy hàm khởi tạo Input của ta sẽ như thế này

```
bool Input::CreateInput()
{
    HRESULT hresult;
    //Khoi tao direct input
    hresult=DirectInput8Create(GetModuleHandle(NULL),     //HINSTANCE : hinstance handle current program
                               DIRECTINPUT_VERSION,     //Direct input version
                               IID_IDirectInput8,        //a reference identifier for the version of Direct input you want to use
                               (void**)&Dinput,         //Con tro tro toi doi tuong direct input (luu y : kieu void)
```

```
                                    NULL);                      //always pass NULL

    //Kiem tra xem Khoi tao Direct input co duoc hay khong
    if(FAILED(hresult))
    {
        MessageBox(0,"Loi xay ra " ,0,0);
        return false;
    }

    //init keyboard
    hresult=m_pInput->CreateDevice(GUID_SysKeyboard,&DIKeyboard,NULL);

    if(FAILED(hresult))
        return false;

    //init Mouse
    hresult=m_pInput->CreateDevice(GUID_SysMouse,&DIMouse,NULL);

    if(FAILED(hresult))
        return false;

    return true;
}
```

Cuối cũng chúng ta cũng khởi tạo xong Direct Input object và Direct Input device , việc tiếp theo ta cần làm là khởi tạo các thuộc tính của mỗi thiết bị để sử dụng chúng , trong hiểu biết của mình thì mình chỉ trình bày cách khởi tạo thuộc tính và sử dụng của Keyboard và Mouse, còn về các loại Joystick thì nếu thích các bạn có thể tự tìm hiểu thêm.

Pasted from <http://gameuit.com/gameuit/showthread.php?295-GameUIT-1st-Challenge-Basic-Direct-Input>

II. Keyboard

1. Khởi tạo thuộc tính và cách sử dụng Keyboard

Mình sẽ xây dựng 1 phương thức để làm công việc này.
Việc đầu tiên ta cần làm là set format cho input device , để DirectX hiểu rằng, input device này đang handle thiết bị nào. Để làm được điều đó ta sử dụng hàm SetDataFormat:

```
hresult = m_pKeyBoardDevice->SetDataFormat(&c_dfDIKeyboard);
```

Để sử set data format cho Mouse thì ta sẽ dùng c_dfDIMouse, và joystick thì dùng c_dfDIJoystick.
Tiếp tục là SetCooperativeLevel , hàm này giúp device của chúng ta sẽ được có ưu thế như thế nào đó

```
hresult=m_pKeyBoardDevice-
>SetCooperativeLevel(hwnd,DISCL_BACKGROUND | DISCL_NONEXCLUSIVE);
```

Tham số đầu tiên là hwnd của chương trình , tham số thứ 2 mới là điều thú vị mà ta cần tìm hiểu. Các loại có thể pass vào tham số thứ 2 này bao gồm :

- **DISCL_BACKGROUND** : Device có thể đọc bất cứ lúc nào, ngay cả khi ứng dụng game không được active
- **DISCL_EXCLUSIVE** : Ứng dụng yêu cầu truy cập độc quyền, không có ứng dụng khác được truy cập vào device trong khi nó đang được sử dụng. Tuy nhiên các ứng dụng nonexclusive khác truy cập vào device vẫn được.
- **DISCL_FOREGROUND** : Game yêu càu device khi mà nó đang active, nếu nó mất focus thì device tạm ngưng hoạt động
- **DISCL_NONEXCLUSIVE** : Ứng dụng này không cần truy cập độc quyền. flag này nói cho các cờ khác đang hoạt động tiếp tục sử dụng device này
- **DISCL_NOWINKEY** : Flag này thông báo DirectInput vô hiệu hóa các phím Windows trên bàn phím.

Các trường hợp xảy ra là

Chúng ta sẽ coi một vài ví dụ như sau, giả sử ta đang có một ứng dụng media nghe nhạc được điều khiển bằng remote. Khi ứng dụng này ở background (có ứng dụng khác dạng NONEXCLUSIVE đang sử dụng), chúng ta bấm remote thì nó vẫn hoạt động, như vậy lúc này nó đang được để DISCL_BACKGROUND. Bây giờ ta xem thêm 1 chút, ví dụ có thêm 1 ứng dụng xem phim sử dụng cùng remote. Như vậy nếu ta bấm play thì cả 2 ứng dụng nhạc và phim cùng chạy, lúc này ta cần thiết lập 2 ứng dụng này flag DISCL_EXCLUSIVE để chỉ 1 trong 2 ứng dụng có thể dùng.

Tiếp theo đó bạn có phải lấy Device để sử dụng bằng hàm Acquire(VOID).
Và tất nhiên, khi đã kết thúc 1 cái gì đó cũng cần giải phóng nó ra, lúc này bạn có thể sử dụng hàm Unacquire(VOID)

Sau đây là code để khởi tạo cho KeyBoard

```
bool Input::InitKeyboard(HWND hwnd)
{
    HRESULT hresult;

    hresult=m_pKeyBoardDevice->SetDataFormat(&c_dfDIKeyboard);

    if(FAILED(hresult))
        return false;

    hresult=m_pKeyBoardDevice-
>SetCooperativeLevel(hwnd,DISCL_BACKGROUND | DISCL_NONEXCLUSIVE);

    if(FAILED(hresult))
        return false;

    DIPROPDWORD dipdw;

    dipdw.diph.dwSize       = sizeof(DIPROPDWORD);
    dipdw.diph.dwHeaderSize = sizeof(DIPROPHEADER);
    dipdw.diph.dwObj        = 0;
    dipdw.diph.dwHow        = DIPH_DEVICE;
    dipdw.dwData            = 256; // Arbitary buffer size
```

```
    hresult = m_pKeyBoardDevice->SetProperty( DIPROP_BUFFERSIZE, &dipdw.diph );
    if (hresult!=DI_OK) return false;

    hresult=m_pKeyBoardDevice->Acquire();

    if(FAILED(hresult))
        return false;

    return true;
}
```

Chắc hẳn bạn sẽ thấy vài dòng lạ, những dòng đó dùng để Set 1 số thuộc tính(properties) cần thiết cho device mà bạn muốn.
Tham khảo thêm
Set any properties of the device that you desire with IDirectInputDevice8:: SetProperty(). This is device context-sensitive, meaning that some devices have some properties and some don't. Thus, you have to know what you're trying to set. In this case, you'll only use this to set some of the range properties of the joystick device, but be aware that in most cases, anything that is configurable on a device is configured with a call to SetProperty(). As usual, the call is fairly horrific, but I'll show you exactly how to do it when we cover the joystick example.
Và trên **MSDN**

Sau khi đã khởi tạo thành công Keyboard device, ta sẽ sử dụng nó bằng cách thiệt lập bộ đệm (buffer) của bàn phím. Và một lúc nào đó, trong vòng lặp game của mình, ta cần phải xem trong bộ đệm có phím nào được nhấn, để làm thế này, ta phải fill các thuộc tính của bàn phím vào bộ đêm bằng hàm GetDeviceState

Trước tiên là cần khai báo bộ đệm cần dùng (ở đây ta dùng bộ đêm 256):

```
class Input
{
//some code...
    char                keys[256];
//some code....
tiếp theo là hàm fill các thuộc tính của bàn phím vào bộ đệm :

PHP Code:
void Input::PollKeyBoard()
{
        m_pKeyBoardDevice->Acquire(); // Acquire device
        m_pKeyBoardDevice-
>GetDeviceState( sizeof(keys), (LPVOID)&keys); // Fill bo dem
                                    //Tham so dau la size cua bo dem
                                    //Tham so thu hai la con tro tro toi bo dem

}
Giờ chỉ cần việc xét xem phím nào được nhấn

PHP Code:
if(keys[DIK_ENTER] & 0x80)
    {
        //do something here
    }
```

Pasted from <http://gameuit.com/gameuit/showthread.php?295-GameUIT-1st-Challenge-Basic-Direct-Input>

III. Mouse

1. Khởi tạo thuộc tính và sử dụng Mouse

Cũng giống như KeyBoard ta cũng phải thiết lập device handle cho thiết bị, SetCooperativeLevel ,...
Tất nhiên cách sử dụng Chuột sẽ khác với Bàn phím. Ta có thể sử dụng 1 trong 2 struct sau (trong header input.h) để xem xét trạng thái của chuột :

```
typedef struct _DIMOUSESTATE {
    LONG    lX;
    LONG    lY;
    LONG    lZ;
    BYTE    rgbButtons[4];
} DIMOUSESTATE, *LPDIMOUSESTATE;
```

hay

```
typedef struct _DIMOUSESTATE2 {
    LONG    lX;
    LONG    lY;
    LONG    lZ;
    BYTE    rgbButtons[8];
} DIMOUSESTATE2, *LPDIMOUSESTATE2;
```

Tùy theo nhu cầu mà bạn sẽ dùng 1 trong 2, ở đây mình sửa dụng loại có 8 button, ta khai báo 1 biến Mouse state như sau

```
class Input
{
//some code...
    DIMOUSESTATE2       m_MouseState;
//some code..
```

Khởi tạo cho Mouse , cũng tương tự như KeyBoard

```
bool Input::InitMouse(HWND hwnd)
{
    HRESULT hResult;

    ZeroMemory(&m_MouseState, sizeof(m_MouseState));

    hResult=m_pMouseDevice->SetDataFormat(&c_dfDIMouse2);
    if(hResult!=DI_OK)
        return false;

    hResult=m_pMouseDevice-
>SetCooperativeLevel(hwnd,DISCL_NONEXCLUSIVE|DISCL_FOREGROUND);
    if(hResult!=DI_OK)
    {
        MessageBox(0,"Error on init Mouse ","Error",MB_OK);
        return false;
    }
```

```
    hResult=m_pMouseDevice->Acquire();

    return true;
}
```

Ta xét các thuộc tính của Mouse , vị trí, button nào được nhấn ... vẫn bằng hàm GetDeviceState

```
void Input::GetMouse()
{
    HRESULT hr=    m_pMouseDevice-
>GetDeviceState(sizeof(DIMOUSESTATE2), (void**)&m_MouseState);
    if(FAILED(hr))
    {
        // Mouse lost, zero out mouse data structure.
        ZeroMemory(&m_MouseState, sizeof(m_MouseState));
        // Try to acquire for next time we poll.
        m_pMouseDevice->Acquire();
    }
}
```

Từ biến kiểu Mouse state bạn có thể lấy được tọa độ X, Y hay Z của chuột , và xét xem button nào được nhấn qua biến

```
BYTE  rgbButtons[8];
```

Để sử dụng ta xây dựng trong class Input các hàm như sau :

```
float Input::mousePosX()
{
    return m_MouseState.lX;
}

float Input::mousePosY()
{
    return m_MouseState.lY;
}

float Input::mousePosZ()
{
    return m_MouseState.lZ;
}
```

```
bool Input::mouseButtonDown(int button)
{
    return ((m_MouseState.rgbButtons[button] & 0x80)!=0); // The high-
order bit of the byte is set if the corresponding button is down.
}
```

Về index của các button các bạn có thể tìm hiểu thêm trên MSDN.

Mình xin kết thúc bài khởi tạo và sử dụng Direct Input ở đây, hi vọng bài tut này giúp bạn phần nào hiểu được cách sử dụng Direct Input. Nếu có gì sai sót mong các bạn góp ý và bổ sung ^^ Thân.

Source Code : **Source**

Phần 7: Algorithm

I. QuadTree

Mục đích của Quadtree

Quadtree dùng để quản lý toàn bộ các đối tượng của bạn trong game, giúp cho việc xử lý va chạm được chính xác và hiệu quả hơn. Quadtree giúp giảm thiểu số lần xét va chạm giữa các đối tượng, giúp tăng tốc độ xử lý của mỗi chu trình game.

Giả sử bạn đang thực hiện một chương trình đơn giản, trong đó có gồm 500 quả bóng di chuyển theo nhiều hướng khác nhau trong màn hình và va chạm với nhau. Vậy bạn sẽ giải quyết bài toán này như thế nào? Bạn sẽ duyệt toàn bộ 500 quả bóng, và xét va chạm với 499 quả bóng còn lại? Điều này thực sự kém hiệu quả, tốn tài nguyên bộ nhớ và không thực tế. Vậy giải pháp là gì?

Sử dụng Quadtree để phân hoạch không gian

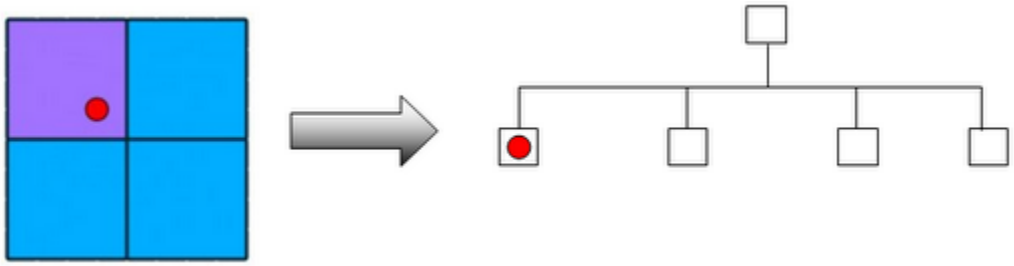
Quadtree giúp bạn nhanh chóng xác định được những đối tượng nào có khả năng xảy ra va chạm, từ đó sử dụng các thuật toán xử lý va chạm phức tạp hơn cho chúng.

Quadtree chia vùng không gian game thành các hình chữ nhật chứa các đối tượng bên trong hình chữ nhật đó. Nó thực hiện điều này bằng cách sau:

Với mỗi node trong Quadtree, node sẽ tiếp tục được chia nhỏ cho đến khi không quá n đối tượng chứa bên trong node đó, hoặc chiều cao cây đạt đến một giới hạn nào đó.

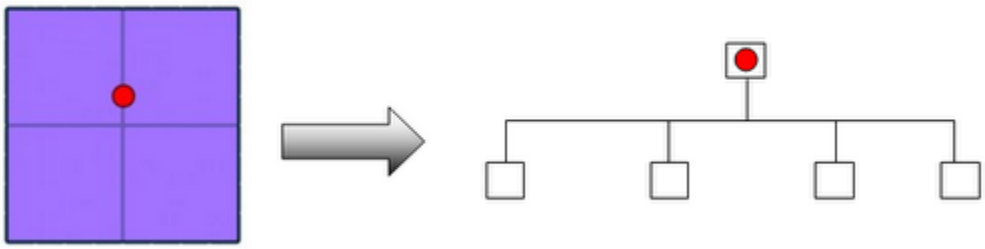
Trong đó n thường nhỏ, 1 hoặc 2.

Khi một node tiếp tục bị chia nhỏ, nó được chia thành 4 hình chữ nhật. Đồng thời, các đối tượng thuộc node mà thuộc một trong 4 hình chữ nhật con đó sẽ được chuyển xuống node con tương ứng của node hiện tại. Xem ví dụ sau:

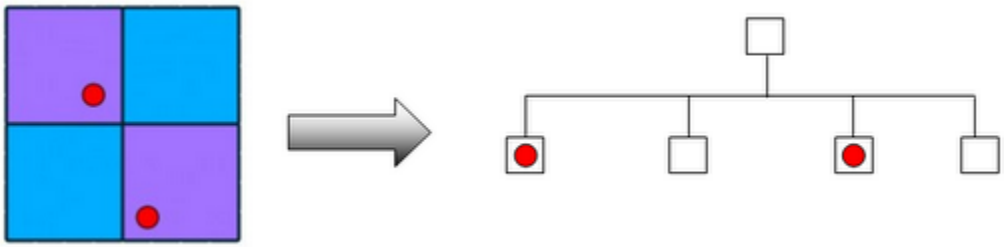


Đối tượng màu đỏ sẽ nằm trên 1 trong 4 node ở mức 2 của QuadTree. (trên

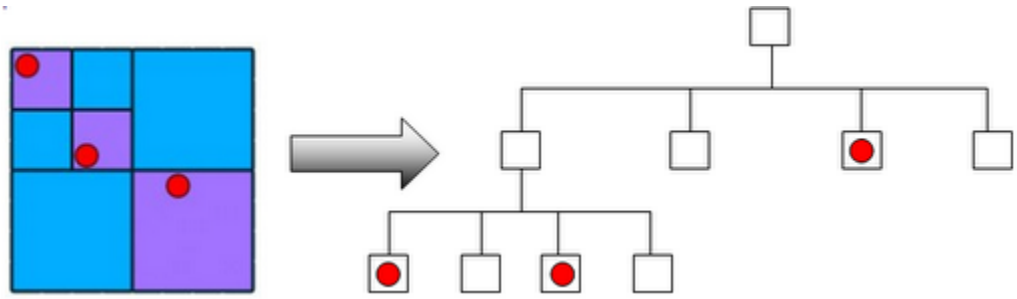
thực tế với trường hợp này, ta chỉ cần lưu đối tượng ở node mức 1, và chỉ chia nhỏ Quadtree sau khi có các đối tượng khác được thêm vào).



Đối tượng sẽ nằm ở node gốc, vì nó không nằm trong bất kỳ hình chữ nhật con nào.



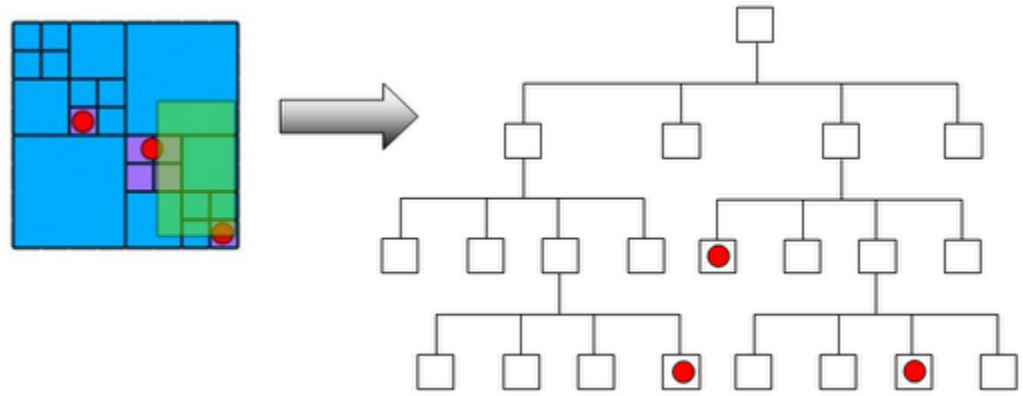
2 đối tượng trên này sẽ nằm ở mức 2 của QuadTree, vì mỗi đối tượng nằm trên 1 node riêng biệt, ta không cần phân chia thêm nữa.



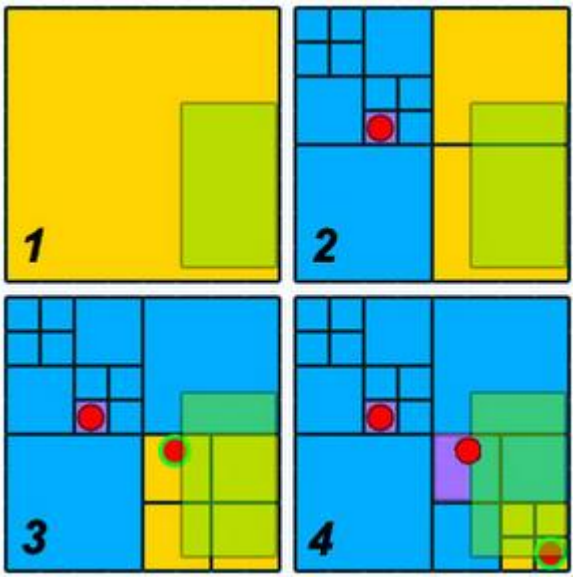
3 đối tượng như hình bên trái sẽ được đưa vào Quadtree. Hình chữ nhật góc trên bên trái có 2 đối tượng nên hình chữ nhật này sẽ được chia nhỏ thêm 1 lần nữa. (đối với Quadtree chứa tối đa 1 đối tượng trong 1 ô).

Vậy ta sử dụng Quadtree vào bài toán xử lý va chạm như thế nào?
Tại mỗi chu kỳ của game, với mỗi đối tượng cần xử lý va chạm, ta kiểm tra Quadtree xem những đối tượng nào có thể xảy ra va chạm với nó, từ đó sẽ tiến hành các thuật toán kiểm tra va chạm phức tạp hơn đối với các đối tượng này.

Xét ví dụ như hình dưới. Ta cần kiểm tra va chạm của hình chữ nhật màu xanh với các đối tượng màu đỏ.



Ta thực hiện các bước kiểm tra sau:



1. Hình chữ nhật xanh va chạm với hình chữ nhật vàng lớn(node mức 1), nhưng không có đối tượng nào ở mức 1.
   2. Hình chữ nhật xanh va chạm với 2 hình chữ nhật vàng nhỏ ở mức 2 (2 node ở mức 2): Cũng không có đối tượng nào. Ta tiếp tục kiểm tra các node con.
   3. Hình chữ nhật xanh va chạm với 4 hình chữ nhật vàng nhỏ ở mức 3 (4 node ở mức 3): có 1 đối tượng, ta đưa đối tượng vào danh sách trả về (danh sách các đối tượng có xảy ra va chạm với hình chữ nhật xanh).
   4. Hình chữ nhật xanh va chạm với 4 hình chữ nhật vàng nhỏ ở mức 4 (6 node ở mức 4): có 1 đối tượng, ta đưa đối tượng vào danh sách trả về.

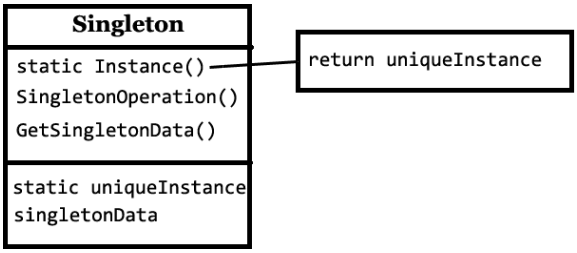
Vậy ta đã có được danh sách các đối tượng xảy ra va chạm với hình chữ nhật xanh.

II. Singleton Pattern

Singleton pattern là 1 design pattern thuộc nhóm thiết lập mẫu (Creational Pattern), cấu trúc của nó cực kì đơn giản vì chỉ bao gồm 1 lớp, lớp này đảm bảo sẽ khởi tạo duy nhất chỉ một phiên bản (instance) của mỗi lớp và cung cấp khả năng truy xuất global tới phiên bản đó (mọi tương tác đều thông qua phiên bản này).
Khi sử dụng Singleton Pattern, đối tượng có thể được sử dụng ở bất kỳ nơi nào trong ứng dụng mà không cần phải khởi tạo contructor.
Singleton Pattern là một giải pháp đặc biệt của Factory Pattern, đối tượng sinh ra là điểm truy cập toàn cục "duy nhất" đối với mọi chương trình gọi đến, hay nói một cách khác tất cả các đối tượng khách gọi đến đều chia sẻ cùng 1 đối tượng được tạo ra, do đó nó tối ưu được bộ nhớ cho game.

Ưu điểm
+ Quản lý việc truy cập tốt hơn vì chỉ có một thể hiện đơn nhất và cho phép cải tiến lại các hoạt động do pattern có thể được kế thừa và tùy biến lại thông qua một thể hiện của lớp con.
+ Người lập trình có thể dùng Singleton Pattern trong những trường hợp chỉ cần một thể hiện duy nhất của một lớp hoặc khi thể hiện duy nhất có thể mở rộng thông qua việc kế thừa, người dùng có thể sử dụng thể hiện kế thừa đó mà không cần thay đổi các đoạn mã của chương trình.
+ Không nên lạm dụng biến static vì nó có thể bị thay đổi bất cứ lúc nào và rất khó cho quá trình debug, thay vào đó, nên sử dụng Singleton Pattern.

Các thành phần
Có 3 yếu tố chính để tạo nên Singleton bao gồm:
+ 1 thuộc tính private static để tham chiếu đến đối tượng duy nhất được tạo ra.
+ 1 hàm khởi tạo private.
+ 1 phương thức public static để truy cập đến instance duy nhất đó nếu nó đã được tạo ra và tạo ra instance duy nhất nếu nó chưa được tạo ra.
Một cách để hạn chế truy cập tới constructor của class thông thường là dùng 1 constructor private.



Hình trên cho thấy cấu trúc của Singleton

Cài đặt

Khởi tạo biến private static _instance dùng để tham chiếu tới đối tượng Singleton được tạo ra.

```
private static Singleton uniqueInstance;
```

Tiếp theo, ta sẽ kiểm tra xem đối tượng đã được khởi tạo chưa thông qua biến _instance, nếu _instance là null thì đối tượng chưa được khởi tạo, do đó, đối tượng sẽ được khởi tạo và được gán tham chiếu là _instance, nếu _instance khác null, tức đối tượng đã được khởi tạo thì ta chỉ trả về tham chiếu _instance của đối tượng đó chứ không khởi tạo thêm đối tượng.

```
public static Singleton Instance()
          {
               if (!uniqueInstance)
               {
                    uniqueInstance = new Singleton();
               }
               return uniqueInstance;
          }
```

Để cấm truy cập đến contructor của 1 class, ta sử dụng hàm khởi tạo private:

```
private Singleton()
          {
          }
```

III. Check Collision

Hiện tại trong các game nhập vai đi cảnh như thế này chưa cần các hàm xử lý va chạm đặc biệt, do đó nhóm chỉ xin giới thiệu tới các bạn phương pháp xét va chạm bằng hình chữ nhật.

Bạn hãy xem những object trong game là những hình chữ nhật. Khi đó việc xét

va chạm giữa các hình chữ nhật là vô cùng đơn giản. Chỉ cần gọi hàm Intersect(…) của class Rectangle.
Khi đã kiểm tra được có va chạm hay không thì bạn cần kiểm tra hướng va chạm. Hướng va chạm rất quan trọng vì nó sẽ quyết định là bạn sẽ xử lý như thế nào đối với hai object va chạm với nhau. Hàm kiểm tra hướng va chạm mà nhóm dùng như sau:

```
public EDirect GetCollisionDir(Rectangle _rect1, Rectangle _rect2)
        {
             if (_rect1.Intersects(_rect2))
             {
                 float top = Math.Abs(_rect1.Top - _rect2.Bottom);
                 float botom = Math.Abs(_rect1.Bottom - _rect2.Top);
                 float left = Math.Abs(_rect1.Left - _rect2.Right);
                 float right = Math.Abs(_rect1.Right - _rect2.Left);
                 float rs = Math.Min(Math.Min(right, left), Math.Min(top,
botom));

                 if (rs == top)
                 {
                     return EDirect.Top;
                 }
                 if (rs == botom)
                 {
                     return EDirect.Bottom;
                 }
                 if (rs == left)
                 {
                     return EDirect.Left;
                 }
                 if (rs == right)
                 {
                     return EDirect.Right;
                 }
             }
             return EDirect.None;
         }
```

Trong đó enum EDirect là enum dùng để lưu giá trị chỉ hướng (tự viết).

Phần 8: Reference

I. Direct Key Code

```
DirectInput Keyboard Scan Codes
DirectX Programming
#ifndef H_DIK
#define H_DIK

// Listed are keyboard scan code constants, taken from dinput.h
```

```
#define DIK_ESCAPE          0x01
#define DIK_1               0x02
#define DIK_2               0x03
#define DIK_3               0x04
#define DIK_4               0x05
#define DIK_5               0x06
#define DIK_6               0x07
#define DIK_7               0x08
#define DIK_8               0x09
#define DIK_9               0x0A
#define DIK_0               0x0B
#define DIK_MINUS           0x0C    /* - on main keyboard */
#define DIK_EQUALS          0x0D
#define DIK_BACK            0x0E    /* backspace */
#define DIK_TAB             0x0F
#define DIK_Q               0x10
#define DIK_W               0x11
#define DIK_E               0x12
#define DIK_R               0x13
#define DIK_T               0x14
#define DIK_Y               0x15
#define DIK_U               0x16
#define DIK_I               0x17
#define DIK_O               0x18
#define DIK_P               0x19
#define DIK_LBRACKET        0x1A
#define DIK_RBRACKET        0x1B
#define DIK_RETURN          0x1C    /* Enter on main keyboard */
#define DIK_LCONTROL        0x1D
#define DIK_A               0x1E
#define DIK_S               0x1F
#define DIK_D               0x20
#define DIK_F               0x21
#define DIK_G               0x22
#define DIK_H               0x23
#define DIK_J               0x24
#define DIK_K               0x25
#define DIK_L               0x26
#define DIK_SEMICOLON       0x27
#define DIK_APOSTROPHE      0x28
#define DIK_GRAVE           0x29    /* accent grave */
#define DIK_LSHIFT          0x2A
#define DIK_BACKSLASH       0x2B
#define DIK_Z               0x2C
#define DIK_X               0x2D
#define DIK_C               0x2E
#define DIK_V               0x2F
#define DIK_B               0x30
#define DIK_N               0x31
#define DIK_M               0x32
#define DIK_COMMA           0x33
#define DIK_PERIOD          0x34    /* . on main keyboard */
#define DIK_SLASH           0x35    /* / on main keyboard */
#define DIK_RSHIFT          0x36
#define DIK_MULTIPLY        0x37    /* * on numeric keypad */
#define DIK_LMENU           0x38    /* left Alt */
#define DIK_SPACE           0x39
#define DIK_CAPITAL         0x3A
#define DIK_F1              0x3B
#define DIK_F2              0x3C
#define DIK_F3              0x3D
#define DIK_F4              0x3E
#define DIK_F5              0x3F
#define DIK_F6              0x40
#define DIK_F7              0x41
#define DIK_F8              0x42
#define DIK_F9              0x43
```

```
#define DIK_F10             0x44
#define DIK_NUMLOCK         0x45
#define DIK_SCROLL          0x46    /* Scroll Lock */
#define DIK_NUMPAD7         0x47
#define DIK_NUMPAD8         0x48
#define DIK_NUMPAD9         0x49
#define DIK_SUBTRACT        0x4A    /* - on numeric keypad */
#define DIK_NUMPAD4         0x4B
#define DIK_NUMPAD5         0x4C
#define DIK_NUMPAD6         0x4D
#define DIK_ADD             0x4E    /* + on numeric keypad */
#define DIK_NUMPAD1         0x4F
#define DIK_NUMPAD2         0x50
#define DIK_NUMPAD3         0x51
#define DIK_NUMPAD0         0x52
#define DIK_DECIMAL         0x53    /* . on numeric keypad */
#define DIK_F11             0x57
#define DIK_F12             0x58

#define DIK_F13             0x64    /*                     (NEC PC98) */
#define DIK_F14             0x65    /*                     (NEC PC98) */
#define DIK_F15             0x66    /*                     (NEC PC98) */

#define DIK_KANA            0x70    /* (Japanese keyboard)            */
#define DIK_CONVERT         0x79    /* (Japanese keyboard)            */
#define DIK_NOCONVERT       0x7B    /* (Japanese keyboard)            */
#define DIK_YEN             0x7D    /* (Japanese keyboard)            */
#define DIK_NUMPADEQUALS    0x8D    /* = on numeric keypad (NEC PC98) */
#define DIK_CIRCUMFLEX      0x90    /* (Japanese keyboard)            */
#define DIK_AT              0x91    /*                     (NEC PC98) */
#define DIK_COLON           0x92    /*                     (NEC PC98) */
#define DIK_UNDERLINE       0x93    /*                     (NEC PC98) */
#define DIK_KANJI           0x94    /* (Japanese keyboard)            */
#define DIK_STOP            0x95    /*                     (NEC PC98) */
#define DIK_AX              0x96    /*                     (Japan AX) */
#define DIK_UNLABELED       0x97    /*                        (J3100) */
#define DIK_NUMPADENTER     0x9C    /* Enter on numeric keypad */
#define DIK_RCONTROL        0x9D
#define DIK_NUMPADCOMMA     0xB3    /* , on numeric keypad (NEC PC98) */
#define DIK_DIVIDE          0xB5    /* / on numeric keypad */
#define DIK_SYSRQ           0xB7
#define DIK_RMENU           0xB8    /* right Alt */
#define DIK_HOME            0xC7    /* Home on arrow keypad */
#define DIK_UP              0xC8    /* UpArrow on arrow keypad */
#define DIK_PRIOR           0xC9    /* PgUp on arrow keypad */
#define DIK_LEFT            0xCB    /* LeftArrow on arrow keypad */
#define DIK_RIGHT           0xCD    /* RightArrow on arrow keypad */
#define DIK_END             0xCF    /* End on arrow keypad */
#define DIK_DOWN            0xD0    /* DownArrow on arrow keypad */
#define DIK_NEXT            0xD1    /* PgDn on arrow keypad */
#define DIK_INSERT          0xD2    /* Insert on arrow keypad */
#define DIK_DELETE          0xD3    /* Delete on arrow keypad */
#define DIK_LWIN            0xDB    /* Left Windows key */
#define DIK_RWIN            0xDC    /* Right Windows key */
#define DIK_APPS            0xDD    /* AppMenu key */

/*
 *  Alternate names for keys, to facilitate transition from DOS.
 */
#define DIK_BACKSPACE       DIK_BACK            /* backspace */
#define DIK_NUMPADSTAR      DIK_MULTIPLY        /* * on numeric keypad */
#define DIK_LALT            DIK_LMENU           /* left Alt */
#define DIK_CAPSLOCK        DIK_CAPITAL         /* CapsLock */
#define DIK_NUMPADMINUS     DIK_SUBTRACT        /* - on numeric keypad */
#define DIK_NUMPADPLUS      DIK_ADD             /* + on numeric keypad */
#define DIK_NUMPADPERIOD    DIK_DECIMAL         /* . on numeric keypad */
#define DIK_NUMPADSLASH     DIK_DIVIDE          /* / on numeric keypad */
#define DIK_RALT            DIK_RMENU           /* right Alt */
```

```
#define DIK_UPARROW         DIK_UP              /* UpArrow on arrow keypad */
#define DIK_PGUP            DIK_PRIOR           /* PgUp on arrow keypad */
#define DIK_LEFTARROW       DIK_LEFT            /* LeftArrow on arrow keypad */
#define DIK_RIGHTARROW      DIK_RIGHT           /* RightArrow on arrow keypad */
#define DIK_DOWNARROW       DIK_DOWN            /* DownArrow on arrow keypad */
#define DIK_PGDN            DIK_NEXT            /* PgDn on arrow keypad */

#endif // H_DIK
```

Pasted from <http://www.gamespp.com/directx/directInputKeyboardScanCodes.html>

II. Virtual key codes

Symbolic constant	Hexadecimal value	Mouse or keyboard equivalent
VK_LBUTTON	01	Left mouse button
VK_RBUTTON	02	Right mouse button
VK_CANCEL	03	Control-break processing
VK_MBUTTON	04	Middle mouse button on a three-button mouse
	0507	Undefined
VK_BACK	08	BACKSPACE key
VK_TAB	09	TAB key
	0A0B	Undefined
VK_CLEAR	0C	CLEAR key
VK_RETURN	0D	ENTER key
	0E0F	Undefined
VK_SHIFT	10	SHIFT key
VK_CONTROL	11	CTRL key
VK_MENU	12	ALT key
VK_PAUSE	13	PAUSE key
VK_CAPITAL	14	CAPS LOCK key
	1519	Reserved for Kanji systems
	1A	Undefined
VK_ESCAPE	1B	ESC key

	1C1F	Reserved for Kanji systems
VK_SPACE	20	SPACEBAR
VK_PRIOR	21	PAGE UP key
VK_NEXT	22	PAGE DOWN key
VK_END	23	END key
VK_HOME	24	HOME key
VK_LEFT	25	LEFT ARROW key
VK_UP	26	UP ARROW key
VK_RIGHT	27	RIGHT ARROW key
VK_DOWN	28	DOWN ARROW key
VK_SELECT	29	SELECT key
	2A	Specific to original equipment manufacturer
VK_EXECUTE	2B	EXECUTE key
VK_SNAPSHOT	2C	PRINT SCREEN key
VK_INSERT	2D	INS key
VK_DELETE	2E	DEL key
VK_HELP	2F	HELP key
	3A40	Undefined
VK_LWIN	5B	Left Windows key on a Microsoft Natural Keyboard
VK_RWIN	5C	Right Windows key on a Microsoft Natural Keyboard
VK_APPS	5D	Applications key on a Microsoft Natural Keyboard
	5E5F	Undefined
VK_NUMPAD0	60	Numeric keypad 0 key
VK_NUMPAD1	61	Numeric keypad 1 key
VK_NUMPAD2	62	Numeric keypad 2 key

VK_NUMPAD3	63	Numeric keypad 3 key
VK_NUMPAD4	64	Numeric keypad 4 key
VK_NUMPAD5	65	Numeric keypad 5 key
VK_NUMPAD6	66	Numeric keypad 6 key
VK_NUMPAD7	67	Numeric keypad 7 key
VK_NUMPAD8	68	Numeric keypad 8 key
VK_NUMPAD9	69	Numeric keypad 9 key
VK_MULTIPLY	6A	Multiply key
VK_ADD	6B	Add key
VK_SEPARATOR	6C	Separator key
VK_SUBTRACT	6D	Subtract key
VK_DECIMAL	6E	Decimal key
VK_DIVIDE	6F	Divide key
VK_F1	70	F1 key
VK_F2	71	F2 key
VK_F3	72	F3 key
VK_F4	73	F4 key
VK_F5	74	F5 key
VK_F6	75	F6 key
VK_F7	76	F7 key
VK_F8	77	F8 key
VK_F9	78	F9 key
VK_F10	79	F10 key
VK_F11	7A	F11 key
VK_F12	7B	F12 key
VK_F13	7C	F13 key
VK_F14	7D	F14 key
VK_F15	7E	F15 key

VK_F16	7F	F16 key
VK_F17	80H	F17 key
VK_F18	81H	F18 key
VK_F19	82H	F19 key
VK_F20	83H	F20 key
VK_F21	84H	F21 key
VK_F22	85H	F22 key (PPC only) Key used to lock device.
VK_F23	86H	F23 key
VK_F24	87H	F24 key
	888F	Unassigned
VK_NUMLOCK	90	NUM LOCK key
VK_SCROLL	91	SCROLL LOCK key
VK_LSHIFT	0xA0	Left SHIFT
VK_RSHIFT	0xA1	Right SHIFT
VK_LCONTROL	0xA2	Left CTRL
VK_RCONTROL	0xA3	Right CTRL
VK_LMENU	0xA4	Left ALT
VK_RMENU	0xA5	Right ALT
	BA-C0	Specific to original equipment manufacturer; reserved. See following tables.
	C1-DA	Unassigned
	DB-E2	Specific to original equipment manufacturer; reserved. See following tables.
	E3 – E4	Specific to original equipment manufacturer
	E5	Unassigned
	E6	Specific to original equipment

		manufacturer
VK_PACKET	E7	Used to pass Unicode characters as if they were keystrokes. If VK_PACKET is used with SendInput, then the Unicode character to be delivered should be placed into the lower 16 bits of the scan code. If a keyboard message is removed from the message queue and the virtual key is VK_PACKET, then the Unicode character will be the upper 16 bits of the lparam.
	E8	Unassigned
	E9-F5	Specific to original equipment manufacturer
VK_ATTN	F6	ATTN key
VK_CRSEL	F7	CRSEL key
VK_EXSEL	F8	EXSEL key
VK_EREOF	F9	Erase EOF key
VK_PLAY	FA	PLAY key
VK_ZOOM	FB	ZOOM key
VK_NONAME	FC	Reserved for future use
VK_PA1	FD	PA1 key
VK_OEM_CLEAR	FE	CLEAR key
VK_KEYLOCK	F22	Key used to lock device

Original equipment manufacturers should make special note of the VK key ranges reserved for specific original equipment manufacturer use: 2A, DBE4, E6, and E9F5.
In addition to the VK key assignments in the previous table, Microsoft has assigned the following specific original equipment manufacturer VK keys.

Symbolic constant	Hexadecimal value	Mouse or keyboard equivalent
VK_OEM_SCROLL	0x91	None
VK_OEM_1	0xBA	";:" for US

VK_OEM_PLUS	oxBB	"+" any country/region
VK_OEM_COMMA	oxBC	"," any country/region
VK_OEM_MINUS	oxBD	"-" any country/region
VK_OEM_PERIOD	oxBE	"." any country/region
VK_OEM_2	oxBF	"/?" for US
VK_OEM_3	oxCo	"`~" for US
VK_OEM_4	oxDB	"[{" for US
VK_OEM_5	oxDC	"|" for US
VK_OEM_6	oxDD	"]}" for US
VK_OEM_7	oxDE	"'" for US
VK_OEM_8	oxDF	None
VK_OEM_AX	oxE1	AX key on Japanese AX keyboard
VK_OEM_102	oxE2	"<>" or "|" on RT 102-key keyboard

For East Asian Input Method Editors (IMEs) the following additional virtual keyboard definitions must be observed.

Symbolic constant	Hexadecimal value	Description
VK_DBE_ALPHANUMERIC	oxofo	Changes the mode to alphanumeric.
VK_DBE_KATAKANA	oxof1	Changes the mode to Katakana.
VK_DBE_HIRAGANA	oxof2	Changes the mode to Hiragana.
VK_DBE_SBCSCHAR	oxof3	Changes the mode to single-byte characters.
VK_DBE_DBCSCHAR	oxof4	Changes the mode to double-byte characters.
VK_DBE_ROMAN	oxof5	Changes the mode

		to Roman characters.
VK_DBE_NOROMAN	0x0f6	Changes the mode to non-Roman characters.
VK_DBE_ENTERWORDREGISTERMODE	0x0f7	Activates the word registration dialog box.
VK_DBE_ENTERIMECONFIGMODE	0x0f8	Activates a dialog box for setting up an IME environment.
VK_DBE_FLUSHSTRING	0x0f9	Deletes the undetermined string without determining it.
VK_DBE_CODEINPUT	0x0fa	Changes the mode to code input.
VK_DBE_NOCODEINPUT	0x0fb	Changes the mode to no-code input.

Original equipment manufacturers should not use the unassigned portions of the VK mapping tables. Microsoft will assign these values in the future. If manufacturers require additional VK mappings, they should reuse some of the current manufacturer-specific and vendor-specific assignments.

See Also

Using Virtual-Key Codes | Manufacturer-specific Virtual-Key Codes

Pasted from <http://msdn.microsoft.com/en-us/library/ms927178.aspx>

Thank you...