Keyboard Input Using DirectInput

Introduction

This is a quick set of instructions for using DirectInput to get basic keyboard input from the user. DirectInput is an alternative to the MFC/Win32 calls for getting keyboard input.

The DirectInput method described here is based on a device-polling model as opposed to the message model that the MFC/Win32 method uses. The primary advantage of this is that you can directly control the keyboard sampling frequency. Ultimately, this will give users a better feeling of control when interacting with your application.

Caveats

The method described here initializes the keyboard device in exclusive mode. This means that no other applications can see what the keyboard is doing while your application in running. Additionally, the MFC portion of YOUR application won't be able to see the keyboard either. This means that tabbing through controls or setting slider values with the arrow keys will not work.

I am by no means a DirectInput expert. There is probably a better way to handle this stuff. If you have any corrections, additions, or suggestions for this tutorial, please let me know.

Step 1 - Link In the DirectInput libraries

First, you need to link in the DirectInput libraries when compiling your program.

Required Libraries: dinput8.lib dxguid.lib

1) In Visual Studio.net, right-click on your Dialog project in the solution explorer and select properties. This will open the project properties dialog box.
2) Select "All Configurations" in the configurations drop-down menu. This will ensure the changes you make will affect both Debug and Release configurations.
3) Select Configuration Properties -> Linker -> Input in the property browser.

4) Enter "dinput8.lib dxguid.lib" in the Additional Dependencies field.
5) Click Okay to save your changes and close the project properties dialog.

Step 2 - Add DirectInput declarations to your Dialog header file

Include the DirectInput header file.

```
#include <dinput.h>
```

Add the following lines of code to your Dialog header file in the private section of your class.

```
// DirectInput Variables
LPDIRECTINPUT8 fDI;                 // Root DirectInput Interface
LPDIRECTINPUTDEVICE8 fDIKeyboard;   // The keyboard device
```

Step 3 - Add initialization code to your OnInitDialog() method

Add the following lines of code to the OnInitDialog() method in your Dialog class implementation file.

```
// --- Start of DirectInput initialization ---

// Create the abstract DirectInput connection
DirectInput8Create(
    GetModuleHandle(NULL),
    DIRECTINPUT_VERSION,
    IID_IDirectInput8,
    (void**)&fDI,
    NULL
);

if (fDI == NULL)
{
    MessageBox("DirectInput Connection Creation Failed!");
    return FALSE;
}

// Create the connection to the keyboard device
fDI->CreateDevice(GUID_SysKeyboard, &fDIKeyboard, NULL);

if (fDIKeyboard)
{
    fDIKeyboard->SetDataFormat(&c_dfDIKeyboard);
    fDIKeyboard->SetCooperativeLevel(
        this->m_hWnd,
        DISCL_FOREGROUND | DISCL_EXCLUSIVE
    );
    fDIKeyboard->Acquire();
```

```
}
else
{
    MessageBox("DirectInput Keyboard initialization Failed!");
    return FALSE;
}

// --- End of DirectInput initialization ---
```

Step 4 - Get the keyboard state and process

Now that you have DirectInput initialized, you want to start processing keyboard input. Since you are polling the keyboard state at regular intervals, I think it makes sense to put this code in the OnTimer() method. This way you are handling keyboard input just before rendering each frame.

1.) Add the following code to the beginning of your OnTimer() method to get the keyboard state.

```
// The following macro allows you to test if
// a key is currently pressed
#define KEYDOWN(name, key) (name[key] & 0x80)

// Here we set up an array of 256 chars. This will hold the
// entire keyboard state after it has been retrieved.
char fDIKeyboardState[256];

HRESULT  hr;

// get the keyboard state
hr = fDIKeyboard->GetDeviceState(
    sizeof(fDIKeyboardState),
    (LPVOID)&fDIKeyboardState
);

if (FAILED(hr))
{
    // It's possible that we lost access to the keyboard
    // Here we acquire access to the keyboard again
    fDIKeyboard->Acquire();
    return;
}
```

2.) Now you need to add code to process the keyboard state as appropriate for your application. You can test to see if specific keys are pressed using the KEYDOWN macro defined in the code above. The macro takes a keyboard state and a DirectInput keyboard constant, and returns true if the key is pressed and false otherwise. A list of the keyboard constants can be found in the DirectX documentation:

DirectInput

-> DirectInput C/C++ Reference

-> Device Constants
-> Keyboard Device

As an example, here is the keyboard handling code from the simple pong game I wrote. This code is in my OnTimer() method after the code listed above, and before I update my world and redraw the Direct3D windows.

```
static D3DXVECTOR3 paddleSpeed(0.0f, 7.0f, 0.0f);
static D3DXVECTOR3 stop(0.0f, 0.0f, 0.0f);

if (KEYDOWN(fDIKeyboardState, DIK_E))
{
    // Red Player Up Pressed
    fWorld.fRedPaddle->SetVelocity(paddleSpeed);
}
else if (KEYDOWN(fDIKeyboardState, DIK_D))
{
    // Red Player Down Pressed
    fWorld.fRedPaddle->SetVelocity(-paddleSpeed);
}
else
{
    // Red Player Nothing Pressed
    fWorld.fRedPaddle->SetVelocity(stop);
}

if (KEYDOWN(fDIKeyboardState, DIK_I))
{
    // Blue Player Up Pressed
    fWorld.fBluePaddle->SetVelocity(paddleSpeed);
}
else if (KEYDOWN(fDIKeyboardState, DIK_K))
{
    // Blue Player Down Pressed
    fWorld.fBluePaddle->SetVelocity(-paddleSpeed);
}
else
{
    // Blue Player Nothing Pressed
    fWorld.fBluePaddle->SetVelocity(stop);
}
```

References

Microsoft DirectX Documentation
Advanced 3-D Game Programming Using DirectX 8.0 by Peter Walsh